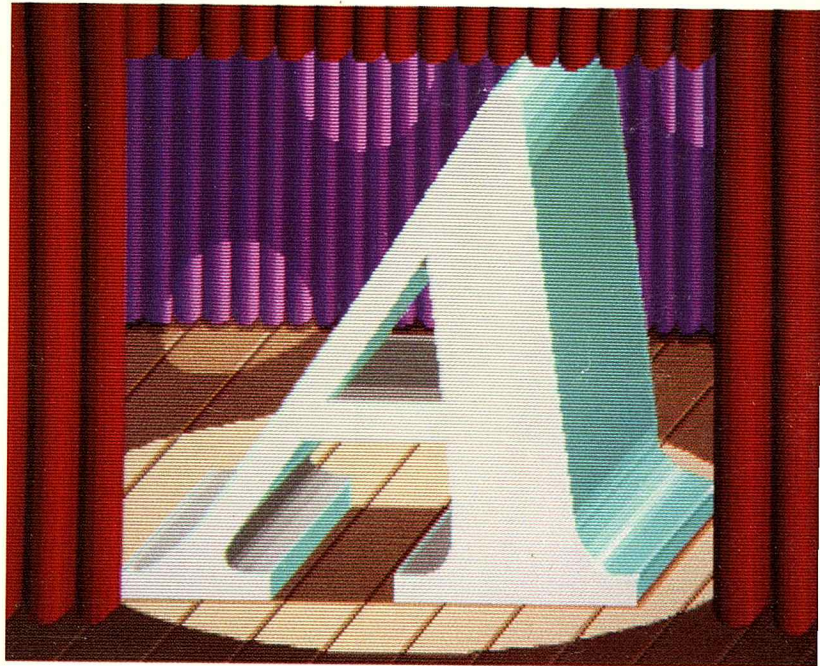


AMIGA™

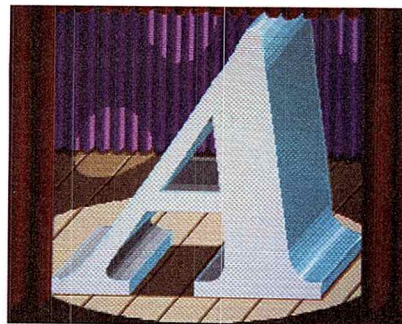
Introduction to

Amiga™



AMIGA

Introduction to
Amiga



COPYRIGHT

This manual Copyright © 1985, Commodore-Amiga, Inc., All Rights Reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from Commodore-Amiga, Inc.

All software described in this manual Copyright © 1985, Commodore-Amiga, Inc., All Rights Reserved. The distribution and sale of these products are intended for the use of the original purchaser only. Lawful users of these programs are hereby licensed only to read the programs, from their media into memory of a computer, solely for the purpose of executing the programs. Duplicating, copying, selling, or otherwise distributing these products is a violation of the law.

DISCLAIMER

COMMODORE-AMIGA, INC. MAKES NO WARRANTIES, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THE PROGRAMS DESCRIBED HEREIN, THEIR QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. THESE PROGRAMS ARE SOLD "AS IS." THE ENTIRE RISK AS TO THEIR QUALITY AND PERFORMANCE IS WITH THE BUYER. SHOULD THE PROGRAMS PROVE DEFECTIVE FOLLOWING THEIR PURCHASE, THE BUYER (AND NOT THE CREATORS OF THE PROGRAMS, COMMODORE-AMIGA, INC., THEIR DISTRIBUTORS, OR THEIR RETAILERS) ASSUMES THE ENTIRE COST OF ALL NECESSARY DAMAGES. IN NO EVENT WILL COMMODORE-AMIGA, INC., BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE PROGRAMS EVEN IF IT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME LAWS 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.

Amiga and Graphicraft are trademarks of Commodore-Amiga, Inc.
Commodore and CBM are registered trademarks of Commodore Electronics Limited.
Alphacom is a registered trademark and Alphapro is a trademark of Alphacom, Inc.
Brother is a registered trademark of Brother Industries, Ltd.
Centronics is a registered trademark of Data Computer Corporation.
Diablo is a registered trademark of Xerox Corporation.
Epson is a registered trademark and FX-80, JX-80, and RX-80 are trademarks of Epson America, Inc.
Hayes is a registered trademark of Hayes Microcomputer Products, Inc.
IBM is a registered trademark of International Business Machines Corporation.
LaserJet and LaserJet PLUS are trademarks of Hewlett-Packard Company.
Okimate 20 is a trademark of Okidata, a division of Oki America, Inc.
Qume is a registered trademark of and LetterPro 20 is a trademark of Qume Corporation.

WARNING: This equipment has been certified to comply with the limits for a Class B computing device, pursuant to subpart J of Part 15 of FCC rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

The last four digits of the Amiga serial number indicate the month and year of manufacture.

Printed in U.S.A.

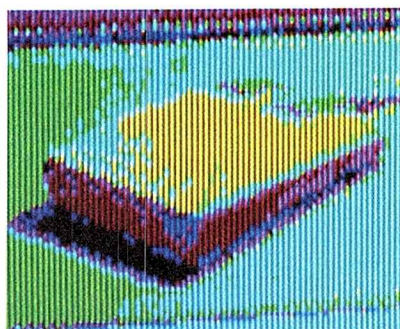
CBM Product Number 372100-01 Rev. C 10-1-85

Contents

Chapter 1:	Introducing the Amiga	1-1
Chapter 2:	Setting Up the Amiga	2-1
Chapter 3:	Getting Started	3-1
Chapter 4:	Using the Workbench	4-1
Chapter 5:	Adding to the Amiga	5-1
Chapter 6:	Caring for the Amiga	6-1
Chapter 7:	Reference	7-1
Appendix A:	Workbench Tools	A-1
Appendix B:	AmigaDOS Messages	A-10
Appendix C:	Changing the Pointer	A-12
Glossary		G-1
Index		I-1



Introducing the Amiga



Welcome! You're about to meet an extraordinary personal computer. It's powerful, yet it's easy to learn and use. It's agile: it can work at several tasks all at the same time. It's colorful and musical. In this manual, you'll meet the Amiga and learn how you can use it both at work and at play.

About This Manual

This manual is for everyone who uses an Amiga. Chapter 2, “Setting Up the Amiga,” shows how to put your Amiga together. If you’re using an Amiga for the first time, read Chapter 3, “Getting Started,” to learn the basics.

Chapter 4, “Using the Workbench,” describes many of the important tasks you perform when using an Amiga.

Chapter 5, “Adding to the Amiga,” describes printers, extra memory, and other add-ons available for your Amiga.

Chapter 6, “Caring for the Amiga,” tells how to keep your Amiga in good working order.

Chapter 7, “Reference,” includes specifications for the Amiga, as well as information about:

- how to change many of the settings for your Amiga
- the Amiga input/output connectors

Appendix A, “Workbench Tools,” tells how to use the Amiga’s built-in clock and notepad, as well as how to see demonstrations of the Amiga’s graphics.

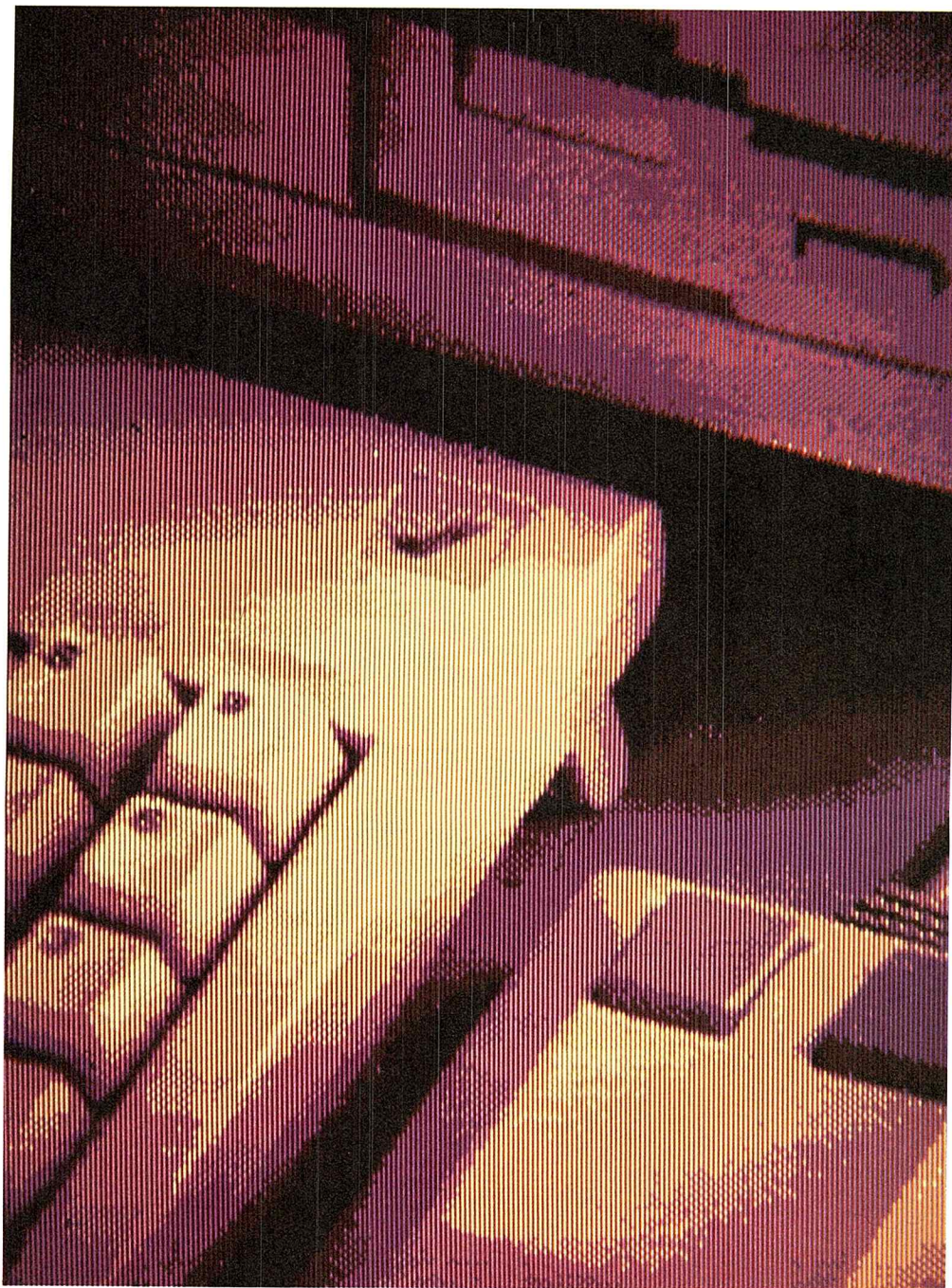
Appendix B, “AmigaDOS Messages,” lists Amiga error numbers and what to do if you get one.

Appendix C, “Changing the Pointer,” shows you how to modify the image that moves on the display when you move the Amiga’s mouse.

At the end, you’ll find a glossary of important terms and an index. Terms from the glossary are shown in *italics* where they first appear in the text.

For More Information

See the *Graphicraft*, *Textcraft*, and *Amiga Basic* manuals to learn how to use these tools. If you are interested in developing software for the Amiga, or if you'd like to learn the details of how the Amiga works, see the *AmigaDOS User's Manual*, the *Amiga Hardware Manual*, the *Amiga ROM Kernel Manual*, the *AmigaDOS Developer's Manual*, the *AmigaDOS Technical Reference Manual*, *The Developer's Guide to the Workbench*, and *Intuition: The Amiga User Interface*. These manuals are available from your Amiga dealer.

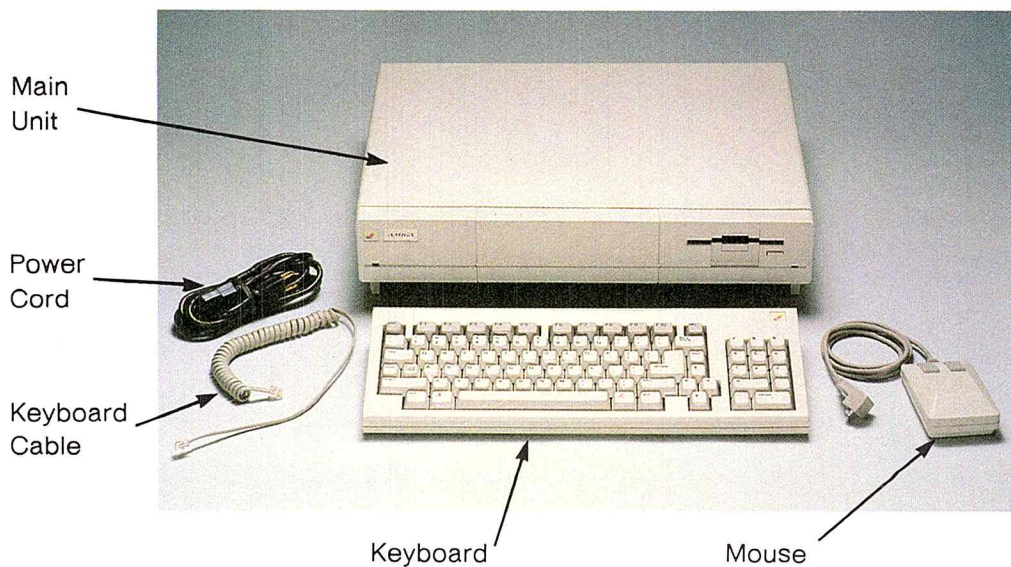


Setting Up the Amiga



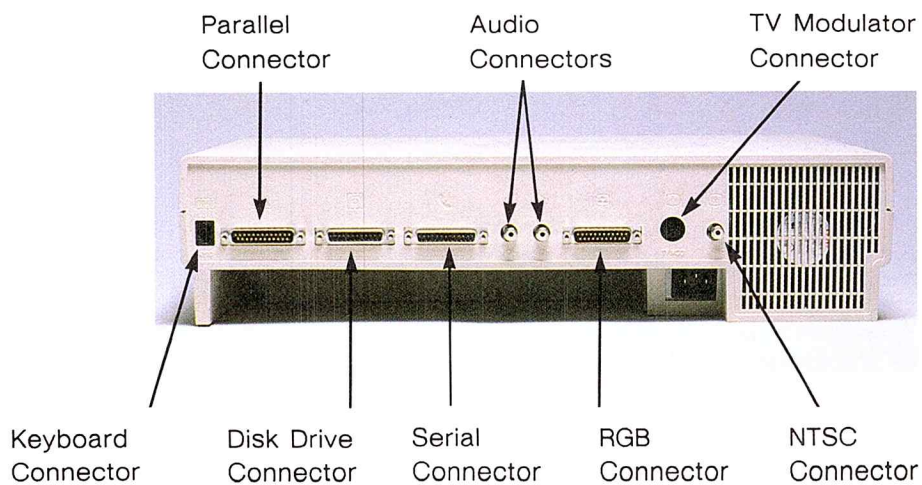
Your Amiga takes only a few minutes to put together. Here are the things you'll find packaged with the Amiga:

- the *main unit*
- the AC power cord
- the *keyboard*
- the *keyboard cable*
- the *mouse*
- three *microdisks*
- manuals, including the one you're reading now
- warranty information



Where to Find the Connectors

On the back of the main unit, you'll find a number of *connectors*. These are places where you attach cables and other devices:



On the right side of the main unit, you'll find two more connectors labeled "1" and "2":



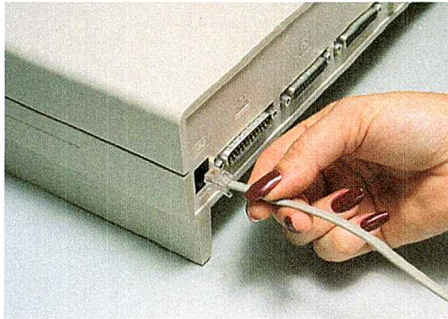
Before You Start

Before putting your Amiga together, **be sure to read each instruction carefully.** While it's not difficult to assemble the Amiga, it *is* possible to damage it if you don't follow the instructions.

When you attach any of the cables or insert anything into a connector, remember: **use a minimum of force.** You can tell when a cable and connector are properly mated when the end of the cable first slides into the connector, then stops when it is in as far as it will go. Always start by pushing gently on the end of a cable. If the cable doesn't slide inward, check the cable and connector to make sure they match and that they're properly oriented before applying more force.

Attaching the Keyboard

Find the keyboard cable, the coiled cable that's straight at one end. Plug the straight end into the *keyboard connector* on the back of the Amiga:



Now put the rest of the keyboard cable underneath the main unit. There's a square opening for the cable to pass through:



Finally, plug the other end of the keyboard cable into the keyboard:



You can change the tilt of the keyboard by folding down the two legs on the bottom. Try both positions to find the tilt that's most comfortable:



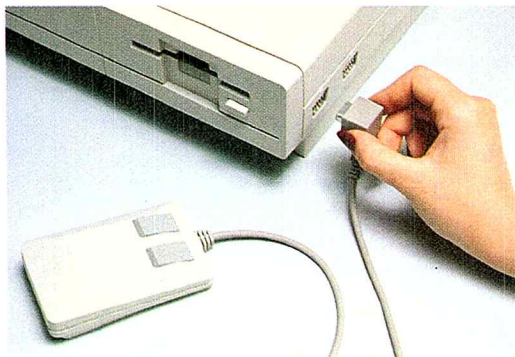
With the legs folded up, you can slide the keyboard under the main unit. This helps save space on your work surface when you're not using the Amiga:



Attaching the Mouse

Before attaching a new mouse, turn the mouse upside down and pull out the piece of foam that holds the *mouse ball* in place. (If you don't get all the foam out, see the "Cleaning the Mouse" section in Chapter 6 to find out how to uncover the mouse ball.)

To attach the mouse, just plug the end of the mouse's cable into the connector labeled "1" on the right side of the main unit. (It's a tight fit; this helps keep the plug in place. Be sure to press firmly.)



If you're right-handed, you'll want the mouse to the right of the keyboard. If you're left-handed, put the mouse to the left. Make sure that the place you set aside for the mouse is at least 12 inches by 12 inches (30 centimeters by 30 centimeters) and that it's clean and flat.

Attaching the Video Monitor

The *video monitor* displays visual information. There are three kinds of video monitors you can use with the Amiga:

- an *RGB monitor*. The *Amiga Monitor* available from Amiga dealers is an RGB monitor. RGB monitors normally produce the best-quality display.
- an *NTSC (composite video) monitor*. There are many kinds of NTSC monitors made specifically for computers. In addition, many newer televisions have NTSC connectors that allow you to connect them directly to computers.
- a television. Conventional televisions (those without NTSC connectors) can also be used as monitors for the Amiga.

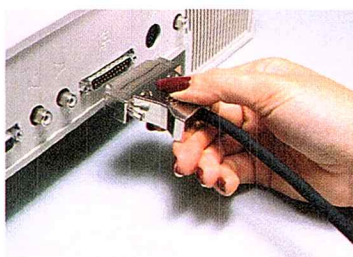
When choosing a monitor, note that televisions and NTSC monitors cannot display as much information as RGB monitors. RGB monitors can show 80 characters clearly on each line on the display, while most televisions and NTSC monitors can show only 60 characters clearly.

Attaching an RGB Monitor

To attach an Amiga Monitor, use the *video cable* supplied with the monitor. Plug the small end of the video cable into the connector on the back of monitor:



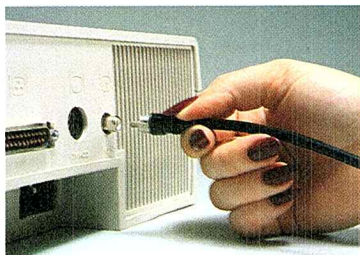
Plug the large end of the cable into the *RGB connector* on the back of the Amiga:



To attach other RGB monitors, see your Amiga dealer for the correct cable and instructions.

Attaching an NTSC Monitor

To attach an NTSC monitor or a television with an NTSC connector to the Amiga, use a shielded cable with a *phono plug* at each end. (You can get this cable from your Amiga dealer.) Plug one end of the cable into the appropriate connector on the monitor, then plug the other end into the NTSC connector on the back of the Amiga:



Attaching a Television

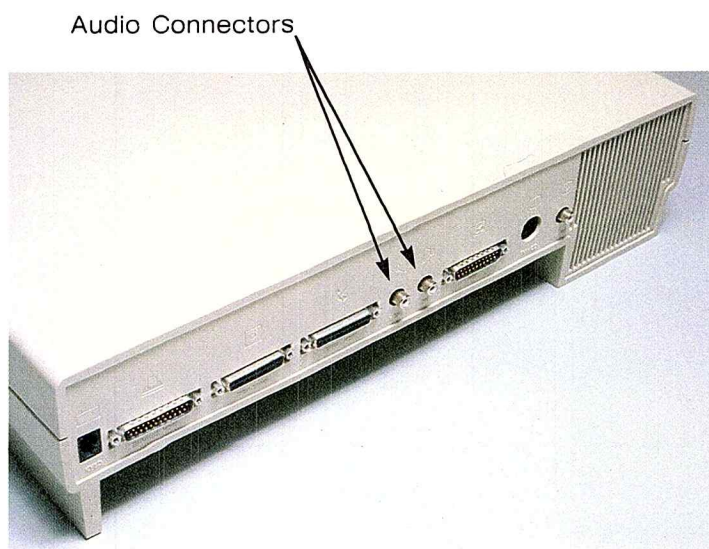
To use a conventional television as a monitor, you need:

- a *TV modulator*
- a *TV modulator cable*
- a *TV switch box*

You can get these from your Amiga dealer. You'll find instructions packaged with them.

Connecting Audio Equipment

The Amiga produces high-quality stereo sound. There are two connectors on the back of the Amiga for connecting the Amiga to audio equipment:



Unless you've attached a conventional television to the Amiga with a TV modulator, you need to connect the Amiga to either an audio system or the audio connector found on some monitors to hear sounds produced by the Amiga.

Connecting the Amiga to a Stereo System

To connect the Amiga to a stereo system, you need two cables. Each cable must on one end have a plug that fits the input connector on your amplifier or receiver (this is most often a phono plug) and on the other end have a phono plug to fit into the Amiga.

On most stereo systems, there are extra inputs, often labeled "Auxiliary" or "Aux," to which you attach one end of each cable. On the back of the Amiga are two *audio connectors*, one for the left audio signal and one for the right audio signal. Insert the other end of each cable into the appropriate audio connector:



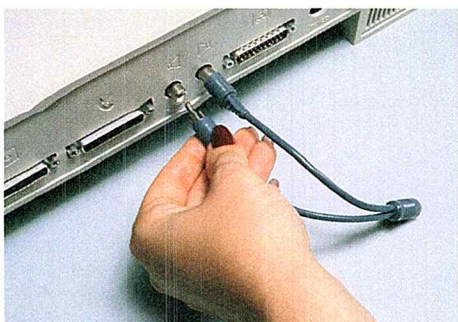
Sound Connections for Monitors

Some monitors, including the Amiga Monitor, have a built-in speaker. To connect a monitor for sound, you need:

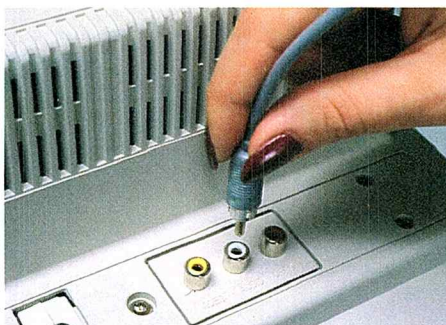
- a "Y" *adapter* that converts the two stereo channels from the Amiga to a single channel
- a cable for connecting the "Y" adapter to the audio connector on the monitor

You can get "Y" adapters and connecting cables from many stores that carry radio and electronic parts.

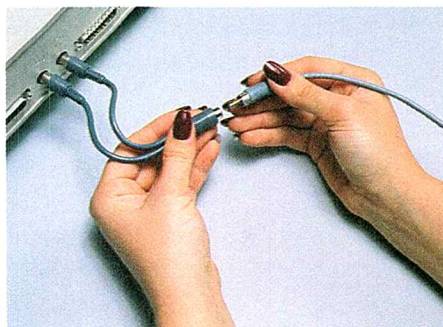
To connect the monitor, insert the two phono plugs at the top of the “Y” into the audio connectors on the back of the Amiga:



Next, insert one end of the connecting cable into the connector on the monitor:

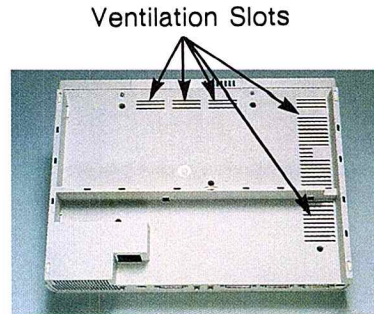
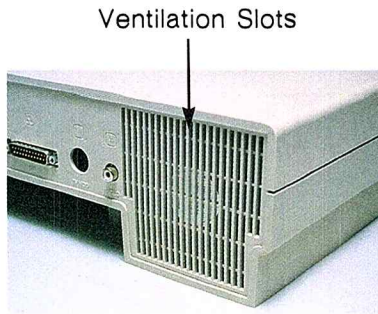


Finally, connect the other end of the cable to the “Y” adapter:



Plugging It In

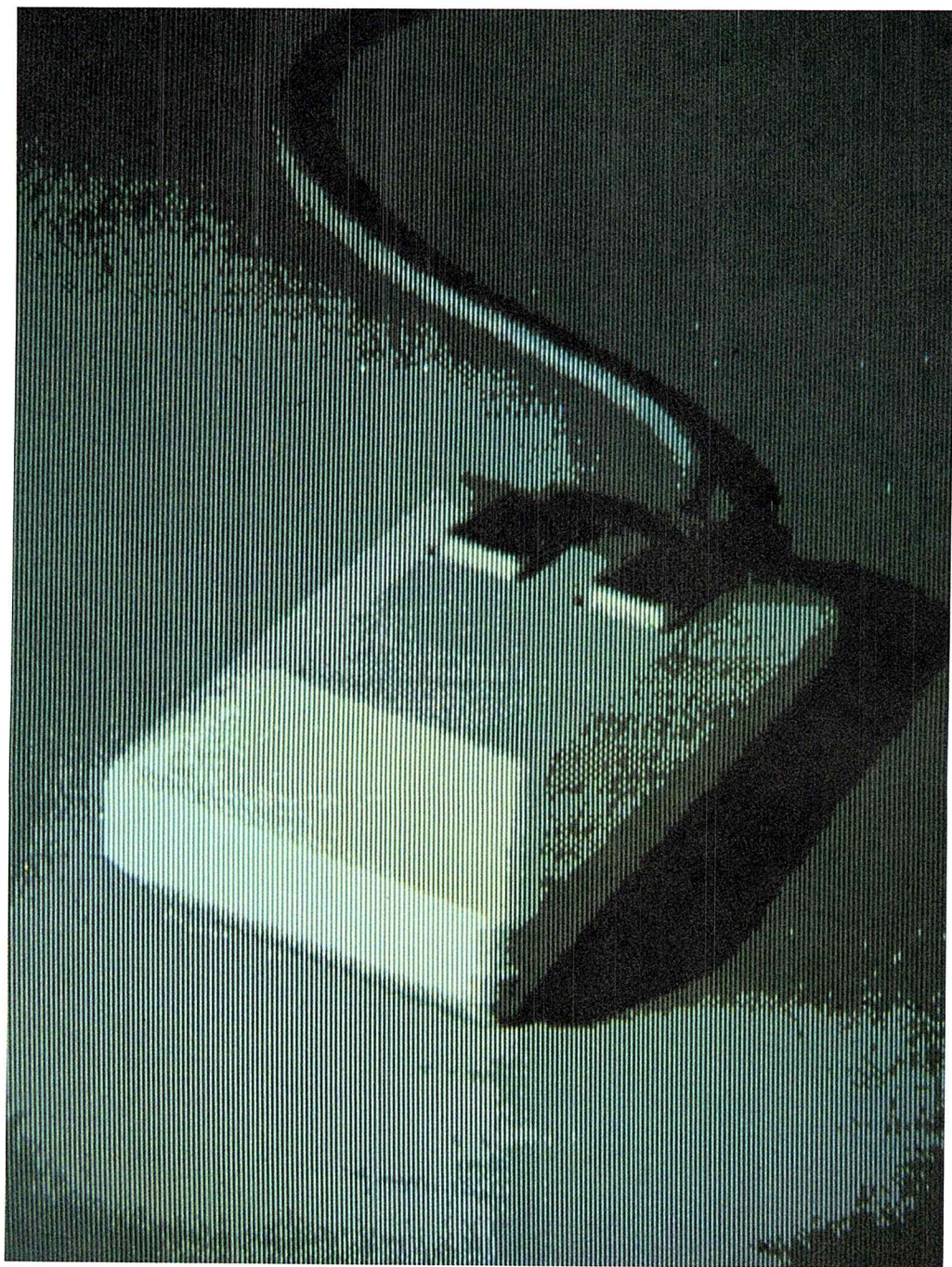
You're almost done putting your Amiga together. Before you plug it in and turn it on, make sure there's nothing covering the ventilation slots on the back or bottom of the main unit:



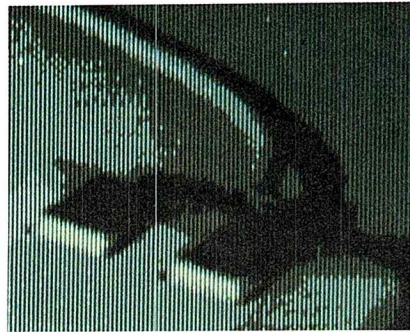
Plug the six-sided end of the power cord into the main unit:



Plug the other end of the power cord into a grounded outlet, and you're ready to start using your Amiga.



Getting Started



In this chapter, you'll learn the basics of using your Amiga. When you're done, you can start using the *tools*, such as the *Graphicraft*™ color graphics tool, that let you work with the Amiga.

A Note About the Mouse

The descriptions in this chapter (and throughout the rest of this manual) assume you're using a mouse. There are, however, certain keys on the keyboard you can use in place of the mouse. To learn how, see the sections "Moving the Pointer Without a Mouse," "Selecting Without a Mouse," and "Using Menus Without a Mouse" in this chapter.

Using Disks

Start by getting the three microdisks—the *Kickstart disk*, the *Workbench disk*, and the *Extras disk*—that came with your Amiga. These disks contain important information used by the Amiga. In addition, have three blank microdisks ready. (You can get blank microdisks from your Amiga dealer.) You'll copy the information from the original disks onto these blank disks and keep the originals in a safe place.

Your original Amiga disks may have *protect tabs*. These are small plastic tabs on the backs of the disks. If you find protect tabs on the original disks, slide each tab toward the edge of the disk until it clicks into place. When you do, you'll be able to see through a small hole in each disk:

Protect Tab



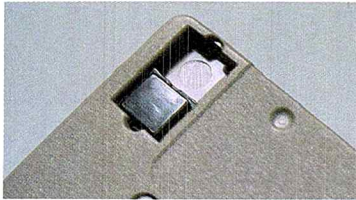
Disk is protected



By putting the protect tabs in this position, you prevent the information on the disks from being changed while they're in the Amiga.

On each of the three blank disks, make sure that the protect tab is toward the middle of the disk, so that it covers the hole. With the tab in this position, you can add new information to a disk:

Disk is unprotected



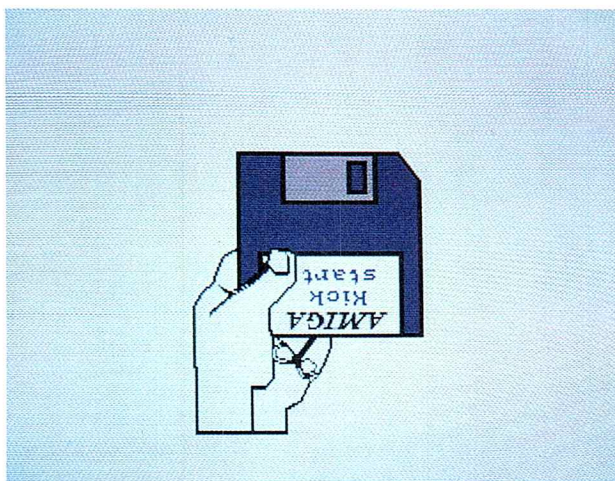
The On/Off switch is on the left side of the main unit. To turn on the Amiga, press the end of the switch labeled "1":



A word of warning:

Whenever you turn off the Amiga, always wait AT LEAST 5 seconds before turning it on again. If you don't observe this precaution, you may damage the Amiga.

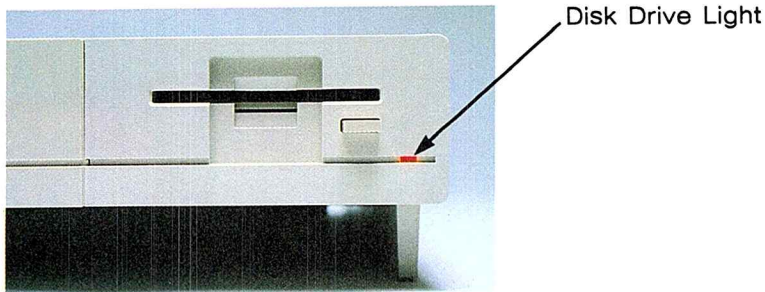
Next, turn on the monitor or television attached to your Amiga. In a few moments, you'll see a picture of a hand holding a Kickstart disk:



This is your cue to insert the Kickstart disk—metal end in, label side up—into the *disk drive*, the device that reads information from disks and adds information to them. Push in the disk until it clicks into place:



After you put in the disk, you'll hear sounds from the Amiga. These are the sounds the disk drive makes as it gathers information. In less than a minute, the Amiga will get the information it needs—with the help of the disk drive—from the Kickstart disk. Notice that while the disk drive is working, the *disk drive light* is on:

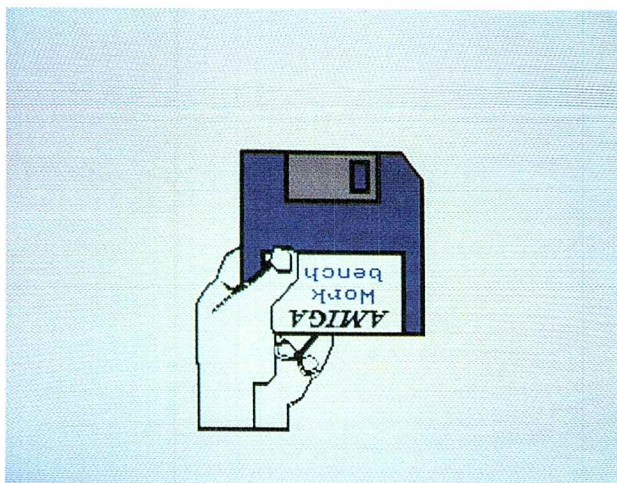


A word of warning:

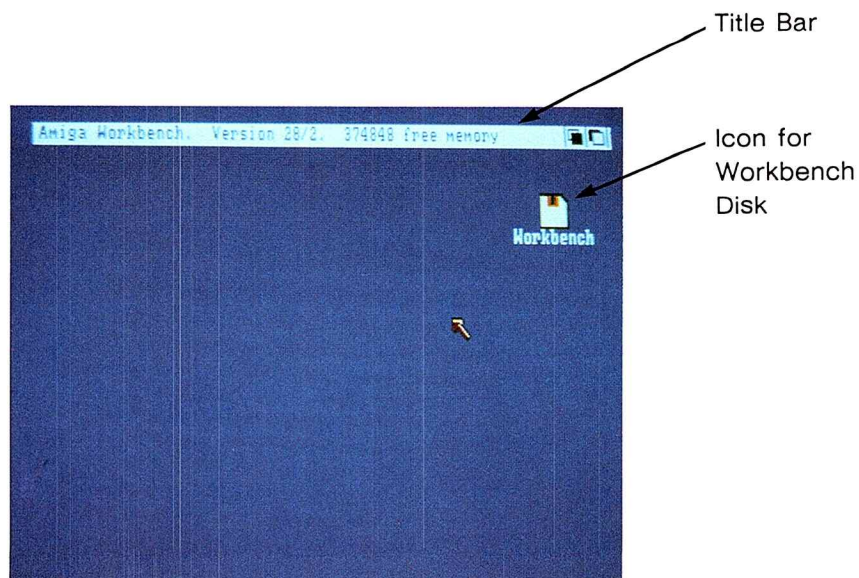
NEVER remove a disk when the disk drive light is on.

The disk drive light tells you that the Amiga is using the disk. Taking a disk out too soon may make it impossible for the Amiga to finish an important task, such as reading the information from the Kickstart disk. Even worse, taking a disk out too early may ruin the information on a disk. Always wait for the disk drive light to turn off before you remove a disk.

When the Amiga is finished with the Kickstart disk, the disk drive light turns off and the hand reappears, this time holding the Workbench disk:



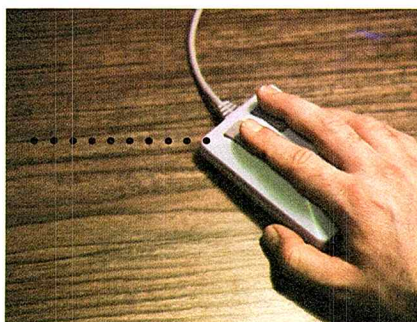
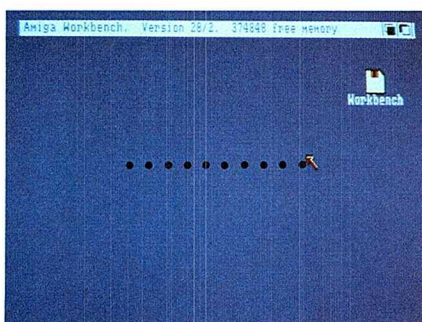
Take out the Kickstart disk by pressing the button on the front of the disk drive, then insert the Workbench disk. In a few moments, you'll see the *Workbench*:



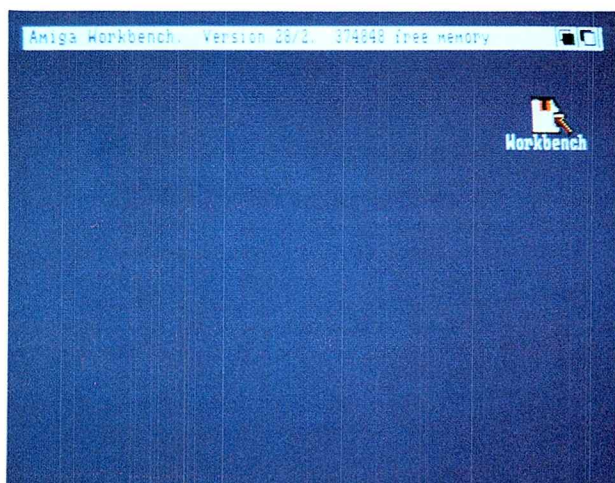
At the top is a *title bar* that identifies the Workbench. On the Workbench is an *icon*, a small picture that represents the Workbench disk. You'll learn more about icons later in this chapter.

Moving the Pointer

You use the *Pointer*, the small arrow on the display, to *point* to the things you want to work with. Moving the mouse moves the Pointer. Without pressing either of the *mouse buttons* on top of the mouse, try rolling the mouse. Be sure to hold the mouse as shown below:



To point, move the Pointer so that its tip is over the thing you want to point to:

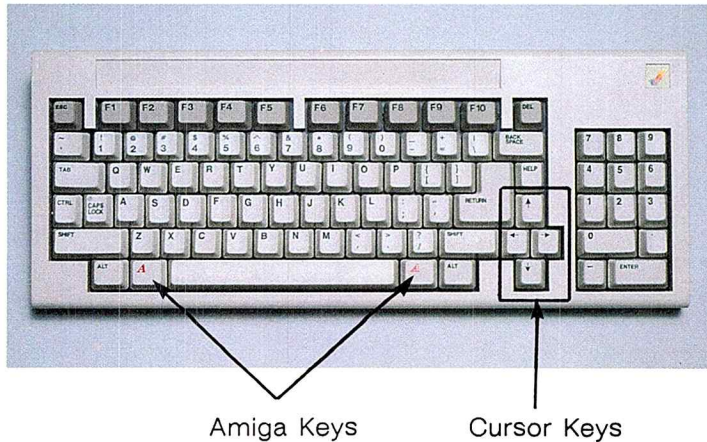


When you first use the mouse, don't worry if it feels a bit clumsy. Once you're familiar with it, you'll find that using the mouse is very quick and convenient.

If you run out of room for your mouse before you get the Pointer where you want it, just lift the mouse and put it down where there's more room. Lifting the mouse doesn't move the Pointer.

Moving the Pointer Without a Mouse

To move the Pointer without a mouse, *hold down* either of the *Amiga keys* while you press a *cursor key*:



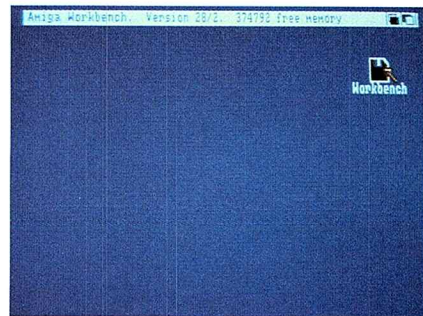
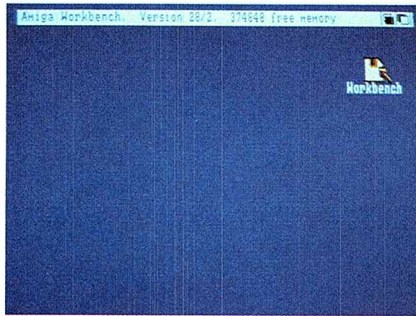
With an Amiga key held down, the Pointer moves in the direction of the arrow on top of the cursor key you press. The longer you hold down the keys, the faster the Pointer moves. To make the Pointer move even faster, hold down both the SHIFT key and an Amiga key while you hold down a cursor key. To stop moving the Pointer, *release* the cursor key.

Selecting Icons

You use the *Selection button*, the left button on the mouse, to *select* icons and other features. Try selecting the icon for the Workbench disk:



Point to the Workbench disk icon, then *click* (*press and quickly release*) the Selection button:



The icon for the Workbench disk is *highlighted* to indicate that it's selected.

Selecting Without a Mouse

To select an icon without using a mouse, first point to the icon, then press both the left Amiga key and the left *ALT* key at the same time:



Left ALT Key

Left Amiga Key

Anything you do by pressing the Selection button on the mouse you can also do by pressing the left Amiga key and the left ALT key at the same time.

Using Menus

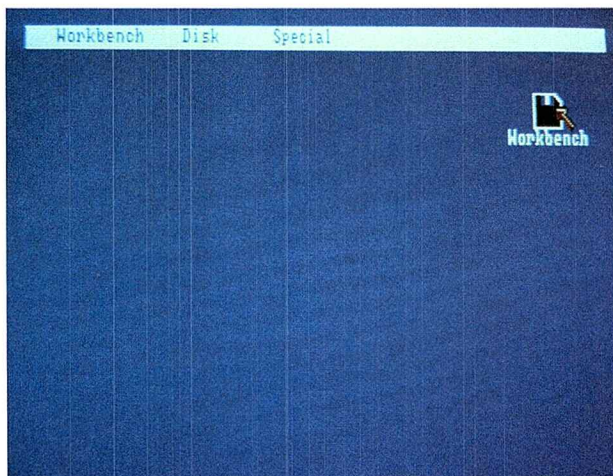
For most tools, including the Workbench, there are *menus* that list choices you can make. To use menus, you use the *Menu button*, the right-hand button on the mouse. The best way to learn how menus work is to try one:



Select the icon for the Workbench disk if it isn't already selected.

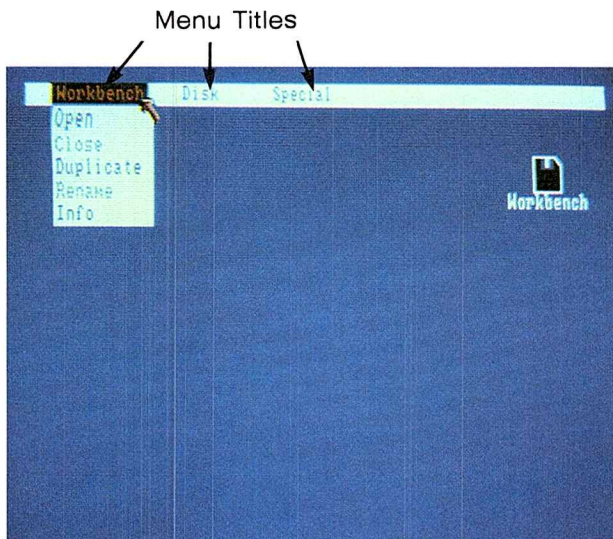


Hold down the Menu button. When you do, the *Menu Bar* appears. In the Menu Bar are *titles* of menus:

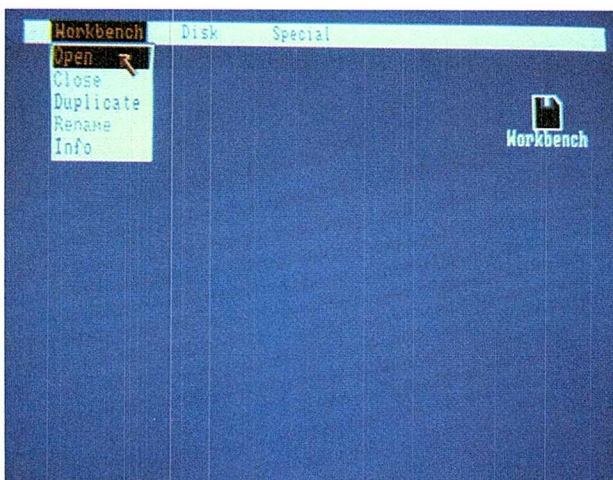




Without releasing the Menu button, point to the title Workbench in the Menu Bar. The *Workbench menu* appears:



While keeping the Menu button down, point to Open in the menu. Open is highlighted:

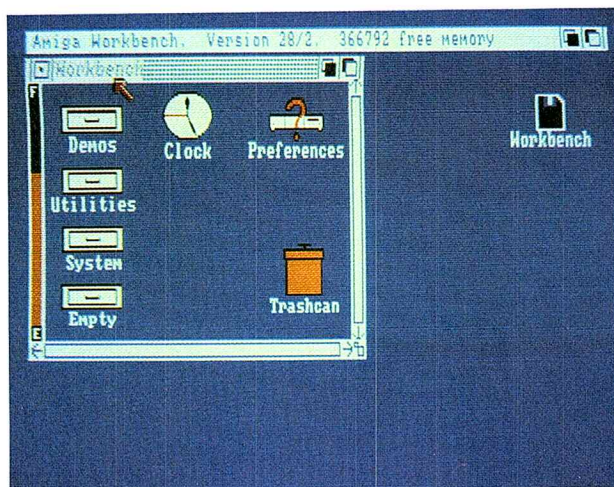




Choose Open by releasing the Menu button while Open is highlighted.

By choosing the Open item from the Workbench menu, you *open* a window for the Workbench disk:

Window for
Workbench
Disk



In the window, you see icons that represent the contents of the Workbench disk.

If you decide you don't want to choose a menu item, move the Pointer off the menu before releasing the Menu button.

To browse through a tool's menus, just hold down the Menu button while moving the Pointer along the Menu Bar. Without choosing an item, you'll get to look at the menu items that are available.

Using Menus Without a Mouse

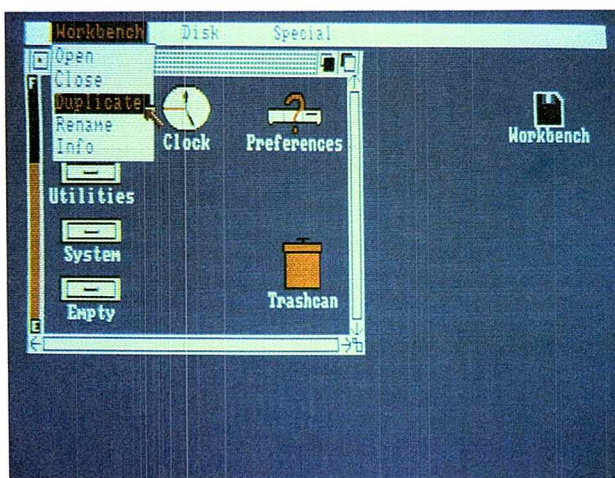
Just as pressing the left Amiga key and the left ALT key at the same time is like pressing the Selection button, pressing the right Amiga key and the right ALT key at the same time is like pressing the Menu button. To use menus without a mouse, hold down the right Amiga key and right ALT key while you move the Pointer with the cursor keys. When the menu item you want is highlighted, release the right Amiga key and right ALT key.

Duplicating Your Disks

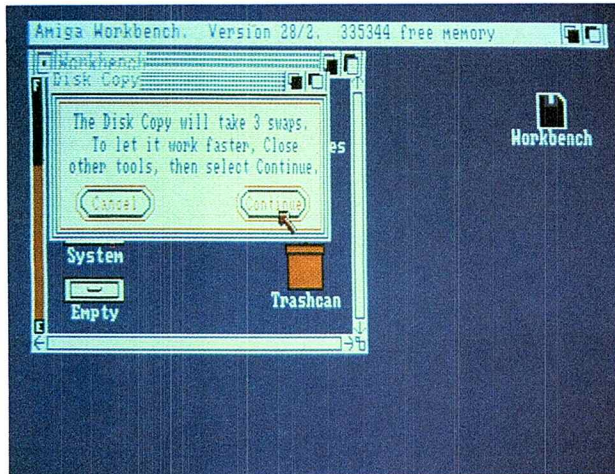
It's important to make duplicates of your original disks and keep the originals in a safe place. You then use the duplicates, called *working disks*, for everyday use. Before you do anything else with the Workbench, follow these directions for duplicating disks:



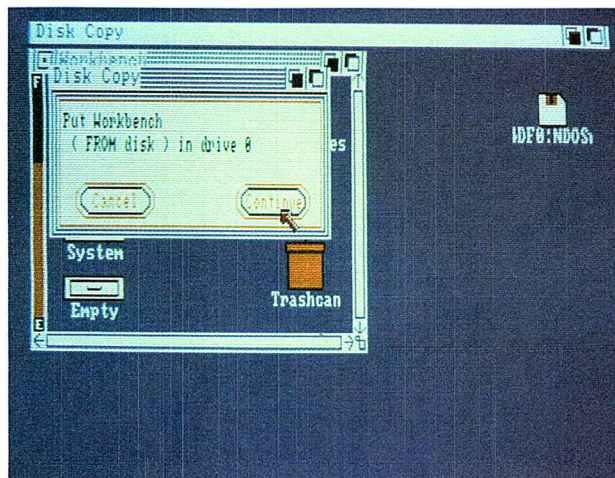
Select the icon for the Workbench disk, then choose Duplicate from the Workbench menu:



A *requester* appears. A *requester* is something the Amiga uses to communicate with you. Here, the requester tells you how many times you'll have to change disks as you copy. Select Continue to go on:



A new requester asks you to put the disk you want to duplicate in drive 0. (Drive 0 is the disk drive in the main unit. Drive 1 is an external disk drive.) Since the Workbench disk is already in drive 0, select Continue to go on:



Next you'll see a requester that asks you to insert the disk to receive the copy. Take out the Workbench disk, insert one of the blank disks, then select Continue.

Finally, there is a series of requesters that ask you to exchange disks. Insert the disk each requester asks for, then select Continue.

When you've finished copying the disk, remove the copy and label it using one of the self-adhesive disk labels packaged with new disks.

Copy the other two disks in the same way: insert the disk you want to copy, select the icon for the disk, choose Duplicate from the Workbench menu, then follow the instructions in the requesters. Be sure to label the copies when you're finished.

When you've made copies of all three disks, put the original disks in a safe place and use only the working disks. That way, if you lose or damage a working disk, you'll be able to make another copy from the original. To learn about proper care for your disks, see Chapter 6, "Caring for the Amiga."

Before going on, you need to insert the newly copied Workbench disk, then *reset* the Workbench. To reset, **make sure the disk drive light is off**, hold down the CTRL key, the left Amiga key, and the right Amiga key at the same time for at least half a second, then release the keys. When you reset the Workbench, you clear the Amiga's *memory*—the electronic circuits the Amiga uses to store information—then the Workbench reappears. You're back to where you were when you first inserted the Workbench disk.

Two warnings:

NEVER reset the Workbench when a disk drive light is on. Resetting when the light is on may damage the information on the disk.

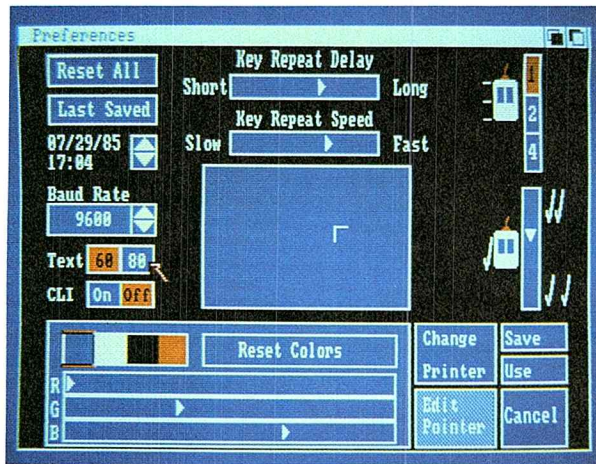
When you reset the Workbench, any work that has not been saved to disk is lost. When you begin using the tools on the Amiga, remember to save your work before you reset.

Using a Tool: Preferences

With the *Preferences* tool, you can make a number of changes to your Amiga. In Chapter 7, you'll find a complete list of Preferences settings. Here, you'll learn how to start using Preferences, how to use Preferences to get the most from your monitor, and what to do when you're finished using Preferences.



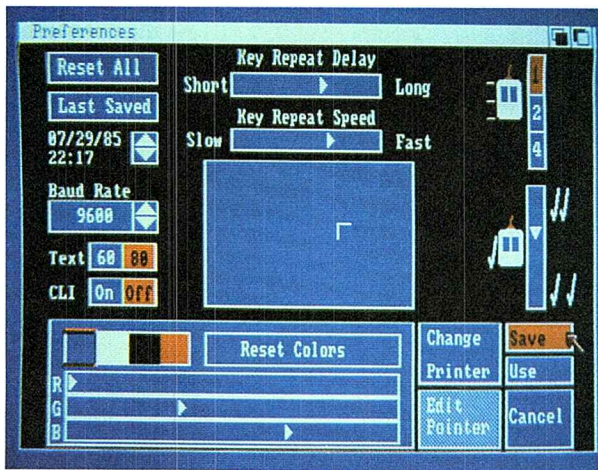
Select the icon labeled "Copy of Workbench," then choose Open from the Workbench menu. When the Preferences icon appears, select it, then choose Open from the Workbench menu. A window for Preferences appears:



Tools use windows to display information and to accept information from you. The Preferences window shows you the current settings for Preferences and lets you change them.

At the left of the Preferences window, you set the number of *characters* (letters, numbers, and symbols) that appear on each line of the display. To the right of the word Text are two *gadgets*, one marked 60 and the other marked 80. If you're using an Amiga Monitor or another RGB monitor, select 80. If you have an NTSC monitor or television attached to your Amiga, select 60.

When you're done, select Save to save your choices on the Workbench disk and return to the Workbench:

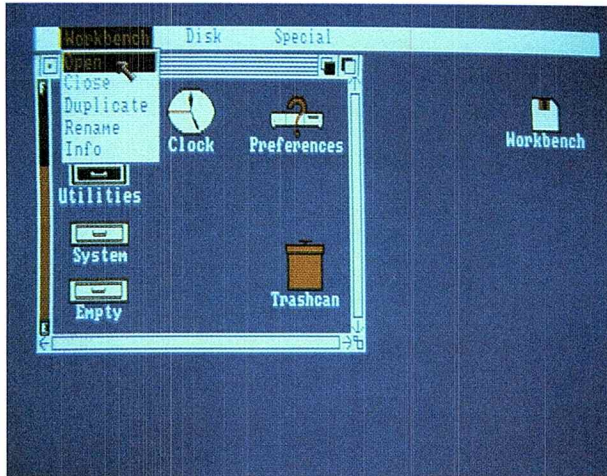


Creating a Project

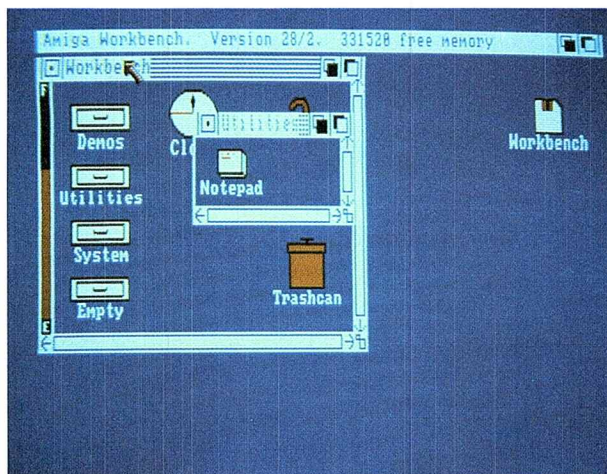
You'll use most Amiga tools to create *projects*. One example of a project is a note you write with the *Notepad*, a tool that is included on your Workbench disk. Here's how to write a note:



Select the Utilities drawer on the Workbench, then choose Open from the Workbench menu:



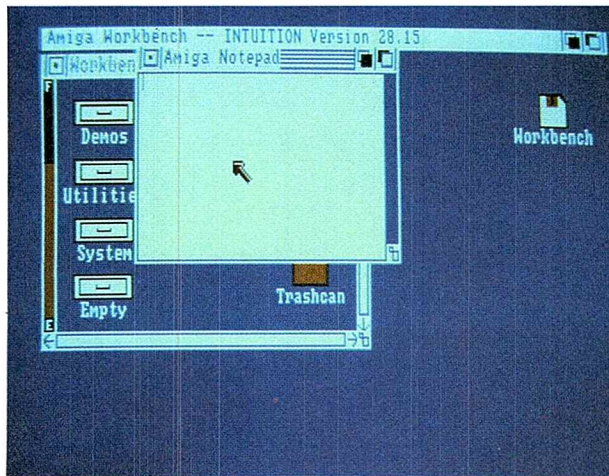
In the window that appears, you'll see the icon for the Notepad:



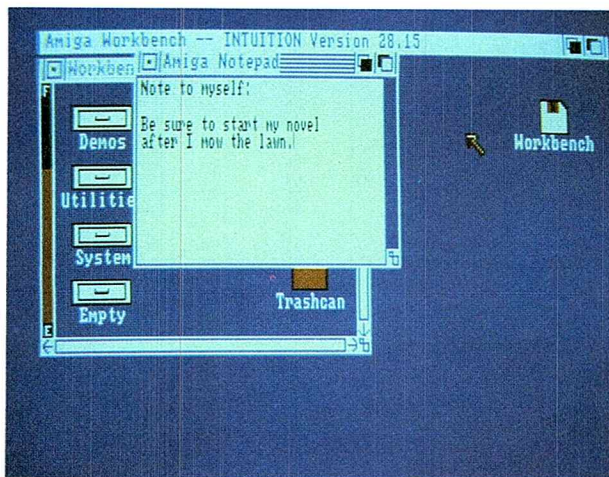
Open the Notepad by selecting its icon, then choosing Open from the Workbench menu.

You can also try another, quicker way to open a tool: point to the icon for the tool, then *double-click* the Selection button. To double-click, quickly press and release the Selection button twice.

In a few moments, a window for the Notepad appears:

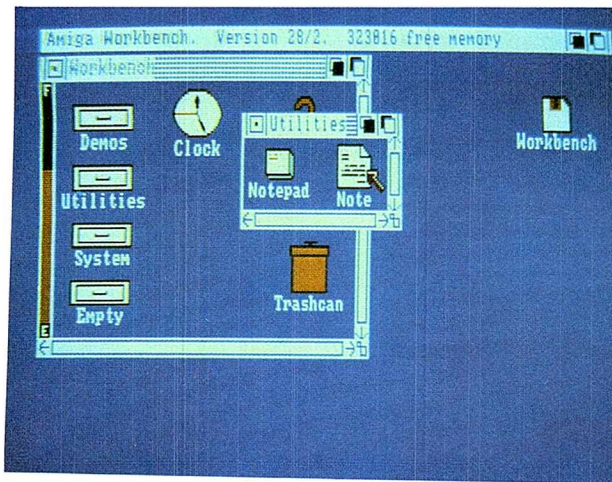


Using the keyboard, type in your note:



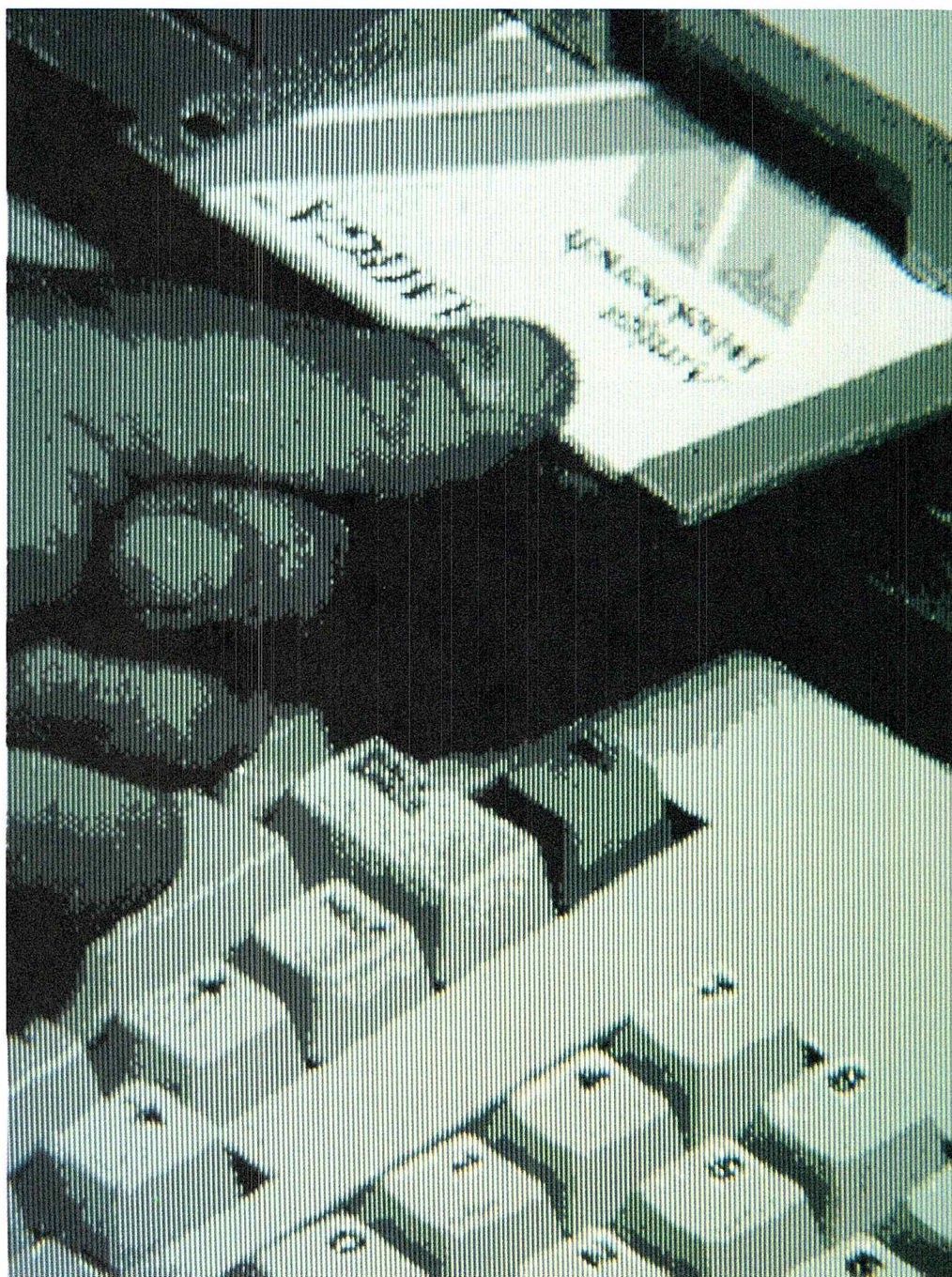
To save your note, choose Save As from the Project menu. (Because the Notepad window is selected, you'll see menus for the Notepad in place of the Workbench menus.) Select the box that appears to the right of the word "Name:", type in a name for your note (the name can be up to 25 characters long), press the RETURN key on the keyboard, then select the OK gadget.

When you're done, choose Quit from the Project menu. The next time you open the Utilities drawer, you'll see a new icon. This is the icon for your note:

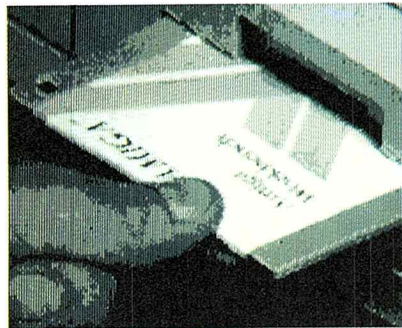


If you want to get back your note, open it by pointing to its icon and double-clicking the Selection button. When you reopen your note, the Notepad is also reopened. You can then add to or change the note.

Now that you're acquainted with the Workbench, menus, and projects, you're ready to use other Amiga tools. Take time now to become familiar with one or more of the tools. When you're done, read Chapter 4 to learn the many other things you can do with the Workbench.



Using the Workbench



The Workbench is a tool you use to control the Amiga. This chapter describes the Workbench and the tasks you perform with it.

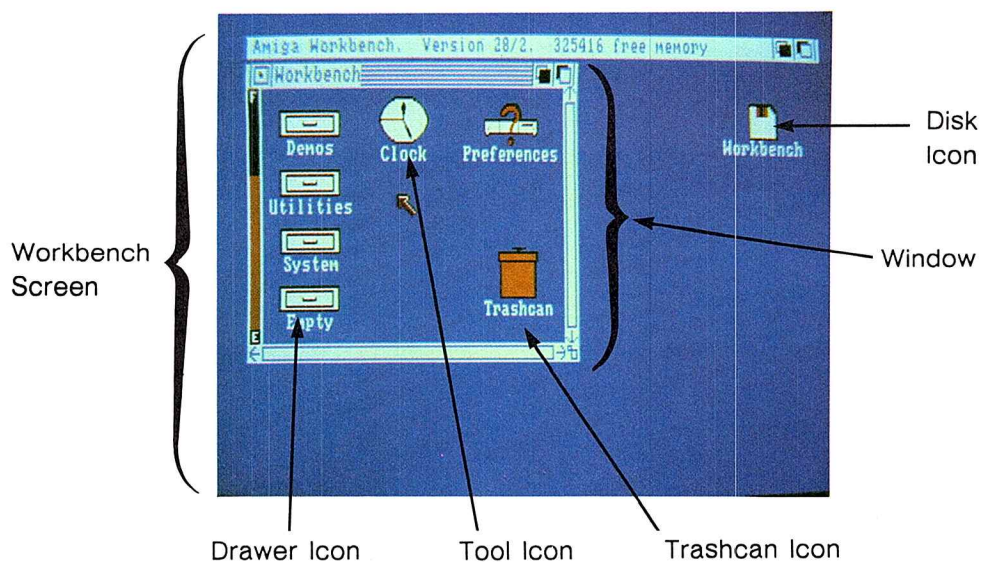
What Is the Workbench?

The Workbench is:

- a tool you use to control the Amiga. You open the Workbench by inserting a disk, called a *Workbench disk*, that contains the Workbench tool.
- an area of the display—a *screen*—set aside for the Workbench.

What's on the Workbench?

When you open the Workbench disk, here are the things you see on the Workbench screen:



Icons

Icons are small pictures that appear on the Workbench. They represent:

- tools
- projects
- disks
- *drawers*, places where you keep tools, projects, and other drawers
- the *Trashcan*, which you use to discard tools, projects, and drawers

Windows

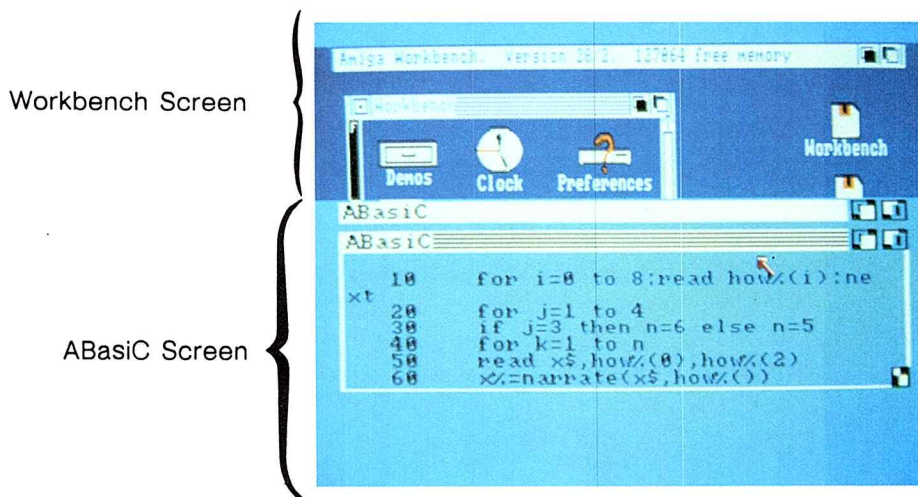
Windows let you see the contents of projects, drawers, disks, and the Trashcan. Each window has a *Title Bar* at the top to identify it. In addition, a window may have one or more *gadgets* that let you change what's being displayed or that let you communicate with a tool. Gadgets are described later in this chapter in the section "Workbench Operations."

Screens

On the Amiga, the way visual information is displayed can be different for different tools. To change the display, tools request different *video attributes*. These attributes include:

- horizontal resolution, the number of *pixels* that appear on each line of the display
- number of colors displayed in the screen
- color palette, the colors that appear in the screen
- interlace, which doubles the number of horizontal lines that appear in the screen

Screens are areas of the display with the same video attributes. They are always as wide as the display. Each screen contains one or more windows:

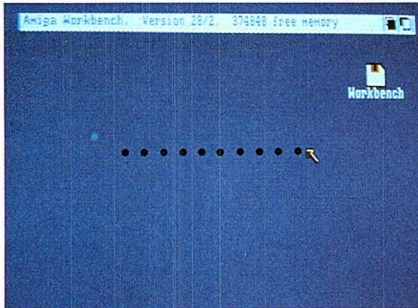


Controlling the Workbench

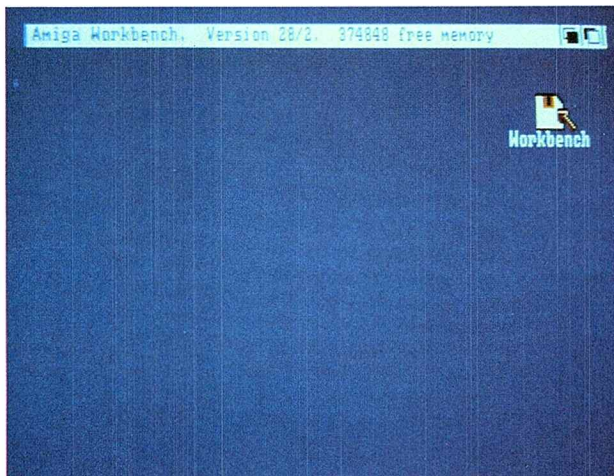
This section explains the techniques you use to perform Workbench tasks. You use many of these same techniques when working with other Amiga tools.

Pointing

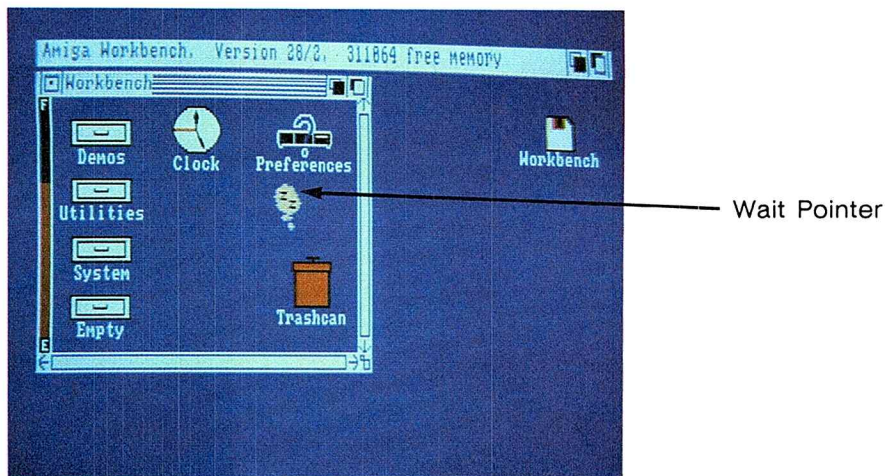
To move the Pointer, you move the mouse:



You *point* to something by moving the Pointer's *point* over it:



There are times when you must wait for the Workbench to finish an activity before you can continue. When this happens, the Pointer changes shape and becomes a *Wait Pointer*:



When the Pointer returns to its original shape, you can continue working.

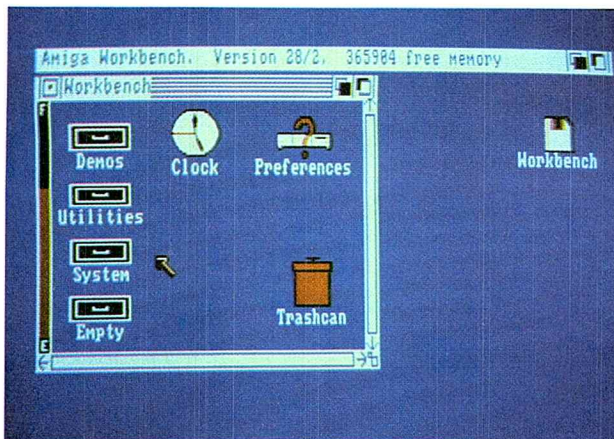
Selecting

To select an icon, point within it, then click the Selection button, the left-hand button on the mouse:



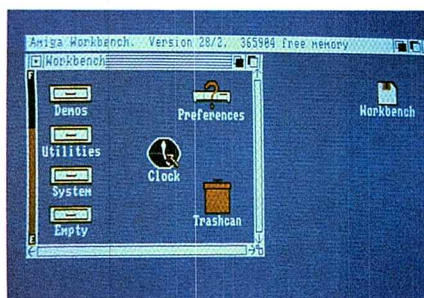
If you no longer want an icon selected, point to a place on the Workbench that isn't occupied by an icon or gadget, then click the Selection button.

Extended Selection is a technique for selecting more than one icon in the same operation. To use it, hold down the SHIFT key while you select icons. Release the SHIFT key when you're done selecting:



Dragging

You move icons, windows, and screens by *dragging* them. To drag an icon, you point to it, hold down the Selection button, and move the mouse. When you hold down the Selection button, the Pointer changes shape:

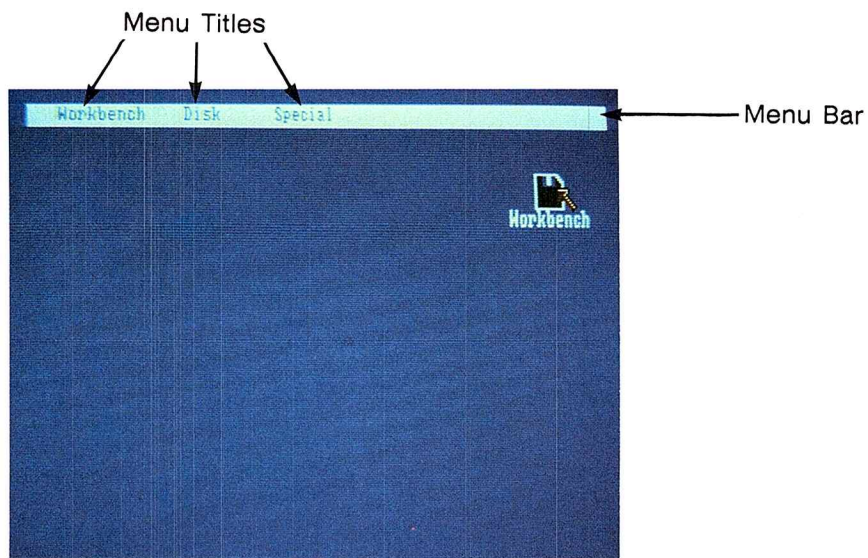


When you release the Selection button, the icon reappears where you've positioned the Pointer.

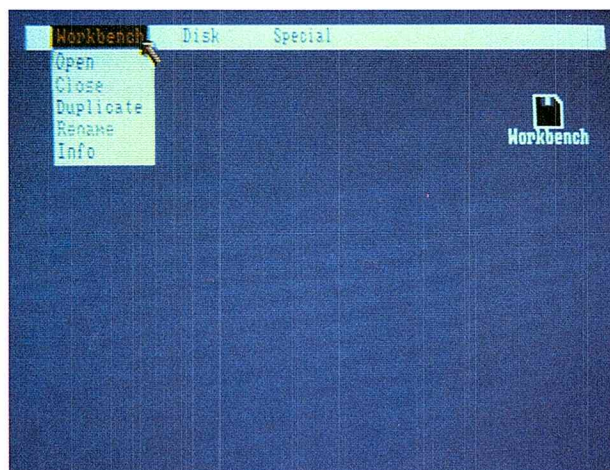
To learn how to drag windows and screens, see the section "Workbench Operations" at the end of this chapter.

Choosing Menu Items

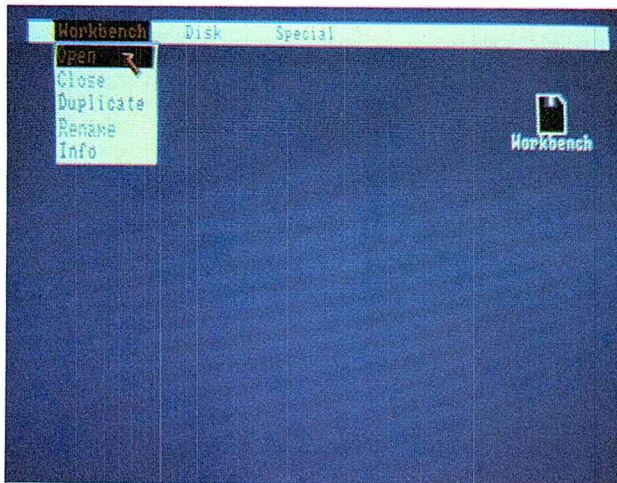
Most tools, including the Workbench, provide menus from which you choose things you can do with the tool. To see the menus that are available, you press the Menu button, the right-hand button on the mouse. When you do, the titles of available menus appear in the Menu Bar, a strip that replaces the Title Bar in the screen in which you're working:



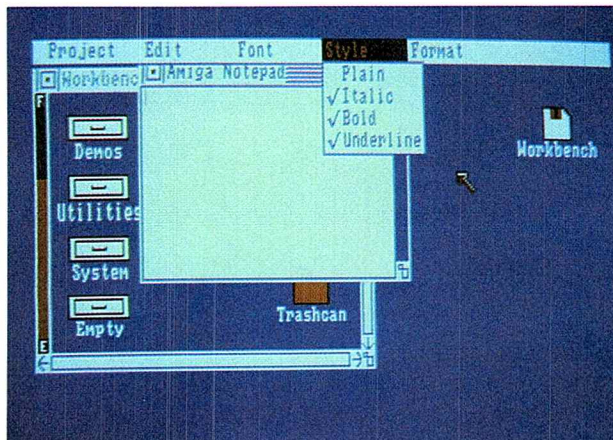
To choose a menu item, hold down the Menu button and move to the title of a menu. The menu appears:



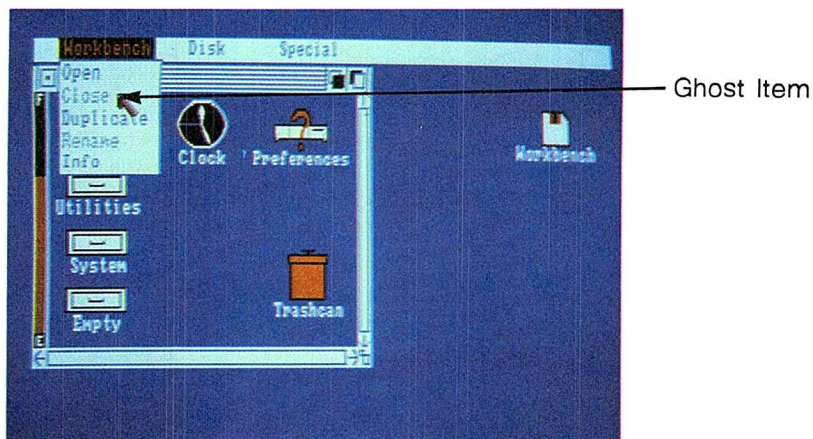
Now point to the item you want to choose and release the Menu button:



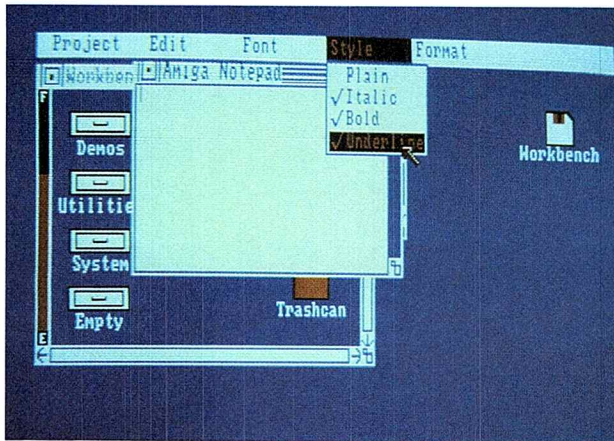
In menus, there are two kinds of items. *Commands* are items that you choose to perform an action. One example of a command is the Open item in the Workbench menu. You choose Open to open a window. *Options* are choices that persist until you choose other, mutually exclusive options. Examples of options are the type styles available in the Amiga Notepad. Options you've chosen are indicated by check marks to the left of the menu items:



In some tools, not all menu items are available at all times. Menu items that you cannot choose appear as *ghost items*:



Multiple Choice is a technique for choosing more than one menu item in the same operation. To use it, hold down the Menu button, then click the Selection button with the Pointer over each of the items you want to choose:



When you're done choosing, release the Menu button.

Shortcuts

A *shortcut* is a quick way, from the keyboard, to select something or to choose a menu item. For a *selection shortcut*, you press a key on the keyboard while holding down the left Amiga key (the key immediately to the left of the Space Bar). For a *menu shortcut*, you press a key on the keyboard while holding down the right Amiga key.

Selection shortcuts for the Workbench are described in the "Workbench Operations" section at the end of this chapter.

Using the Amiga Without a Mouse

On the Amiga, anything you can do with the mouse you can also do from the keyboard:

- To move the Pointer, press an Amiga key and one of the cursor keys (the keys with arrows on top that are to the right of and slightly below the RETURN key) at the same time. This moves the Pointer in the direction of the arrow on the cursor key. The longer you hold down these keys, the faster the Pointer moves.
- To move the Pointer faster, press an Amiga key, the SHIFT key, and one of the cursor keys at the same time.
- Instead of pressing the Selection button (the left button on the mouse), you can press the left Amiga key and the left ALT key (the key just to the left of the left Amiga key) at the same time.
- Instead of pressing the Menu button (the right button on the mouse), you can press the right Amiga key and the right ALT key (the key just to the right of the right Amiga key) at the same time.

Workbench Operations

Using the techniques described in the last section, you can use the Workbench to work with tools, projects, drawers, and disks. This section describes the fundamental Workbench operations.

Operations Involving Tools and Projects

Opening Tools and Projects

When you open a tool or project, you open a window that lets you see the contents of the project or that lets you communicate with the tool. There are two ways to open a tool or project:

- Select the icon for the tool or project, then choose Open from the Workbench menu.
- Point to the icon, then double-click the Selection button.

Opening a project automatically opens the tool used to create it.

On the Amiga, you can have more than one tool open at the same time. This ability is called *multitasking*: the Amiga is able to perform several tasks at once. Note, however, that each new tool you open requires a certain amount of *memory*. Memory is the set of electronic circuits within the Amiga used to keep information. If, when you try to open an additional tool, there isn't sufficient memory for it, the Workbench gives you the message "Cannot open [name of the tool]. Error 103" at the top of the screen.

Duplicating Tools and Projects

Duplicating a tool or project means to make an identical copy in the drawer in which the tool or project resides. To duplicate, select the icon for the tool or project, then choose Duplicate from the Workbench menu.

The name of the new tool or project is "copy of" added to the name of the tool or project that was copied. For example, duplicating the Clock gives you a new tool named "copy of Clock."

Renaming Tools and Projects

To rename a tool or project, select its icon, then choose **Rename** from the **Workbench** menu. When the message appears in the **Title Bar**, select the gadget in the middle of the display, type in the new name, then press the **RETURN** key.

About String Gadgets. The gadget that appears is called a *String Gadget*. (The term *string* refers to a set of one or more characters.) As you use the Amiga, you'll find that String Gadgets appear in requesters when a tool needs information in the form of text. When you use String Gadgets, note that you can change the string that appears in the gadget. Press the **DEL** key to delete the characters at and to the right of the *Text Cursor* (the marker that appears in the gadget). Press the **BACKSPACE** key to delete characters to the left of the text cursor. You can erase what appears in the gadget by pressing the right Amiga key and the **X** key at the same time. You can get back what was in the gadget before you made any changes by pressing the right Amiga key and the **Q** key at the same time.

Getting Information About Tools and Projects

To get information about tools and projects, select the icon for the tool or project, then choose **Info** from the **Workbench** menu. The information includes the *type* (project, tool, drawer, or disk) of the object you've selected, as well as various measures of its size (note that some of these are only of interest to software developers and others who must deal with the inner workings of the Amiga). You can also change the *status* of the object by selecting one of the gadgets below the word **STATUS**. The normal setting is **DELETABLE**, which allows you to delete an object. Select **NOT DELETABLE** if you want to prevent an object from being deleted.

Discarding Tools and Projects

To discard a tool or project, drag its icon over a **Trashcan** icon. When you do, the tool or project is kept in a special drawer maintained by the **Trashcan**. It remains in this drawer until you select the **Trashcan** icon, then choose **Empty Trash** from the **Disk** menu. If you haven't emptied the trash since you last put something in the **Trashcan**, you can retrieve what you discarded by opening the **Trashcan** in the same way you open a project, then dragging its icon to an open drawer somewhere on the **Workbench**. Note that

when you discard something, you do not reclaim disk space until you choose Empty Trash.

You can also discard a tool or project by selecting its icon, choosing Discard from the Workbench menu, then selecting the Retry gadget in the requester that appears. **WARNING:** When you choose Discard to discard a tool or project, you cannot get the tool or project back.

Operations Involving Drawers

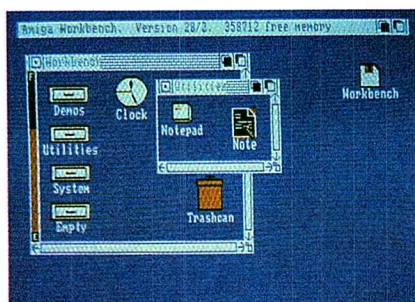
Drawers are places where you can keep tools, projects, and other drawers. You can use drawers to keep order on the Workbench and to keep related items together.

Opening Drawers

You open a drawer in the same way you open a tool or project: either point to the icon for the drawer and double-click the Selection button or select the icon for the drawer, then choose Open from the Workbench menu. Opening a drawer gives you a window in the Workbench screen.

Moving Tools, Projects, and Drawers

To move a tool, project, or another drawer into a drawer, open the drawer into which you want to put the tool, project, or drawer, then drag the icon into the drawer's window:



Another way to move a tool, project, or another drawer into a drawer is to drag the icon over the icon for the drawer into which you want to put it.

Duplicating Drawers

To duplicate a drawer, select the icon for the drawer, then choose Duplicate from the Workbench menu. A new drawer, whose name is “copy of” added to the name of the drawer that was duplicated, appears in the window.

To create a new drawer, you duplicate another drawer. The quickest way is to duplicate the empty drawer that appears on the Workbench, then give the new drawer a new name.

Renaming Drawers

To rename a drawer, select the icon for the drawer, then choose Rename from the Workbench menu. A message then appears asking you for a new name. Select the window that appears, type in a name, then press the RETURN key.

Discarding Drawers

To discard a drawer, drag the icon for the Drawer over the Trashcan icon or choose Discard from the Workbench menu. Note that putting a drawer in the Trashcan does not free disk space until you choose Empty Trash from the Disk menu.

You can also discard a drawer by selecting the icon for the drawer, choosing Discard from the Workbench menu, then selecting Retry in the requester that appears. **WARNING: When you choose Discard to discard a drawer, you cannot get the drawer back.**

Special Drawers: Disks and the Trashcan

Disks and the Trashcan are special kinds of drawers. Disks differ from other drawers in these ways:

- You cannot discard a disk by dragging its icon over the Trashcan icon.
- You cannot move a disk into another drawer.

The Trashcan differs from other drawers in these ways:

- You cannot move the Trashcan into another drawer.
- You cannot discard the Trashcan.
- You can remove the contents of the Trashcan by choosing Empty Trash from the Disk menu.

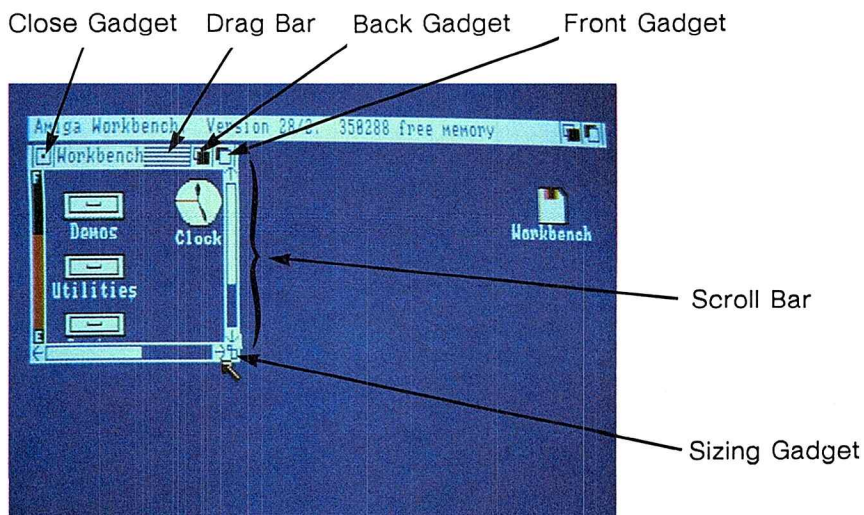
Operations Involving Windows

When you open a tool, project, drawer, disk, or the Trashcan, a window appears on the Workbench. This newly opened window appears in front of any other windows with which it overlaps.

Windows appear within screens. They cannot be moved from one screen to another. While all the windows in a screen can display information, only one window can accept information from you at a time. This window is called the *selected window*.

To select a window, point anywhere within the window and click the Selection button.

You change the size of a window, change what's displayed in the window, move the window, and do other things with windows with the help of *gadgets*. You also use gadgets to communicate with tools. Here are some common gadgets found in windows:



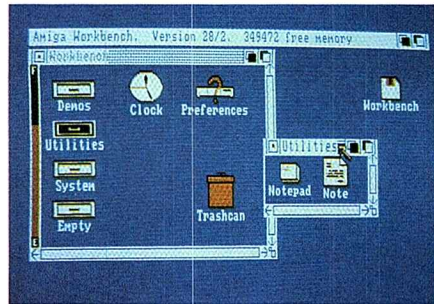
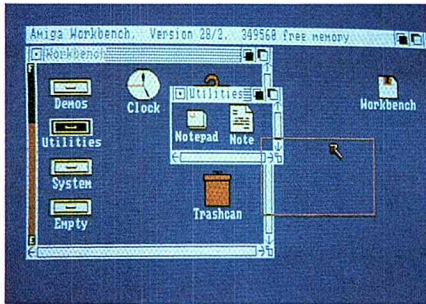
Windows can contain all, some, or none of these gadgets. In addition, windows can contain other gadgets needed for a particular tool.

Like menu items, gadgets in a window can appear as *ghost gadgets*. Here, a ghost Drag Bar indicates that the window is not selected:



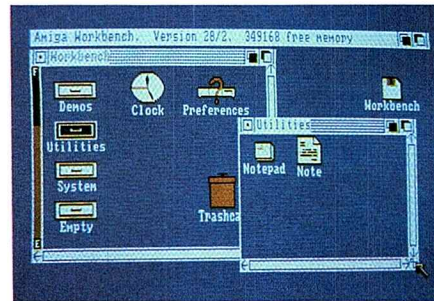
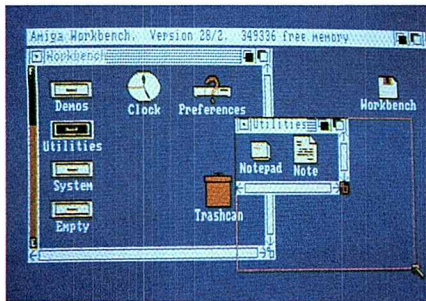
Dragging Windows

You drag a window by pointing anywhere in the window's Title Bar that is not occupied by other gadgets (the *Drag Bar*), holding down the Selection button, and moving the mouse:



Sizing Windows

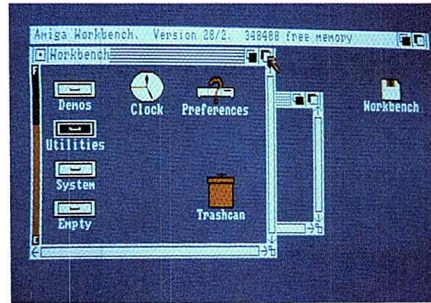
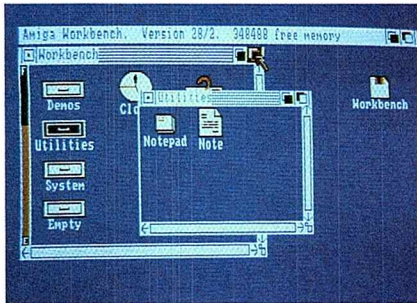
To change the size of a window, you drag its *Sizing Gadget*:



Note that some windows have a maximum size that is smaller than the screen in which they reside.

Moving Windows in Front of Other Windows

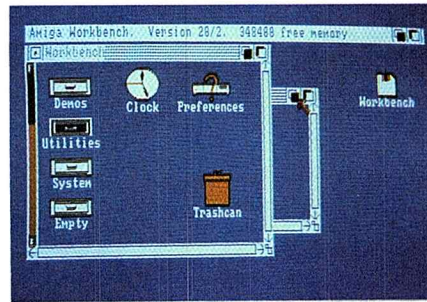
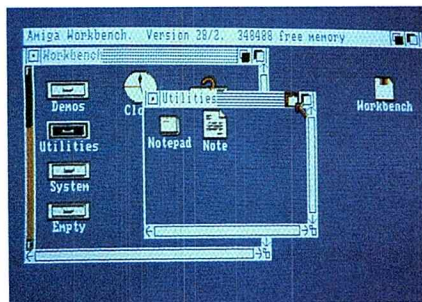
When windows overlap, one window appears in front of the others. To move a window in front of other windows, select the *Front Gadget*:



You can also move a window to the front by pointing to the icon you selected to open the window, then double-clicking the Selection button.

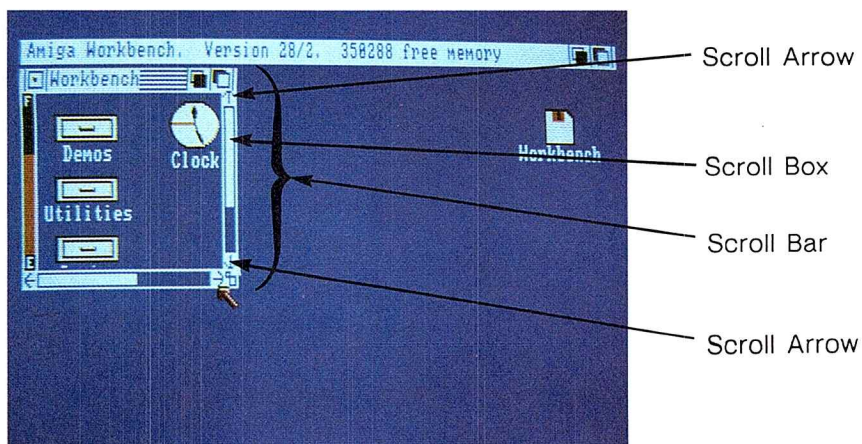
Pushing Windows Behind Other Windows

To move a window behind other windows with which it overlaps, select the *Back Gadget*:

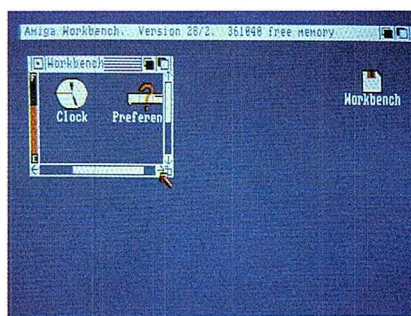


Scrolling the Contents of a Window

For many windows, you can't display everything that can appear within the window at once. Because of this, windows often have *Scroll Bars* that let you move what appears in the window:

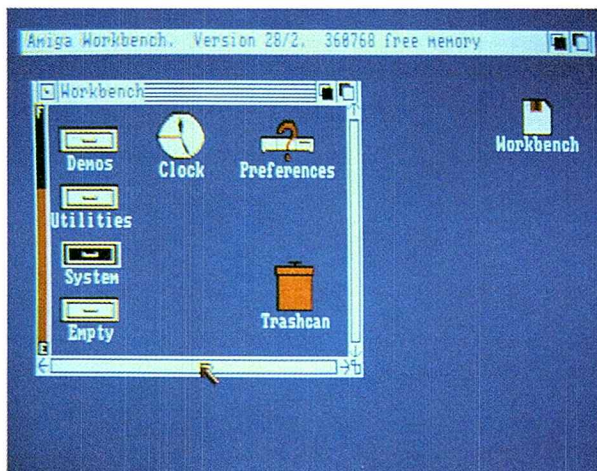


You can move half a window at a time by selecting a *Scroll Arrow* at either end of the Scroll Bar:

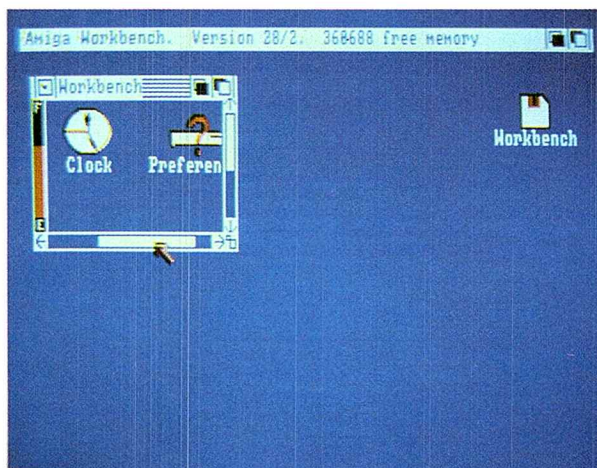


Pressing the Shift Key while selecting a Scroll Arrow moves the window one pixel.

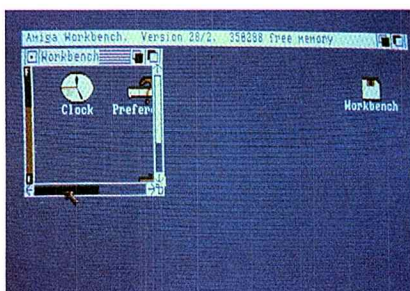
Scroll Boxes change size according to how much can appear in a window. If the window is as wide as what can appear, the Scroll Box in the horizontal Scroll Bar fills the entire space between the Scroll Arrows:



If, for example, only half of what can appear is within the window, the Scroll Box fills only half the space between the Scroll Arrows. The position of the Scroll Box indicates what part you're seeing:



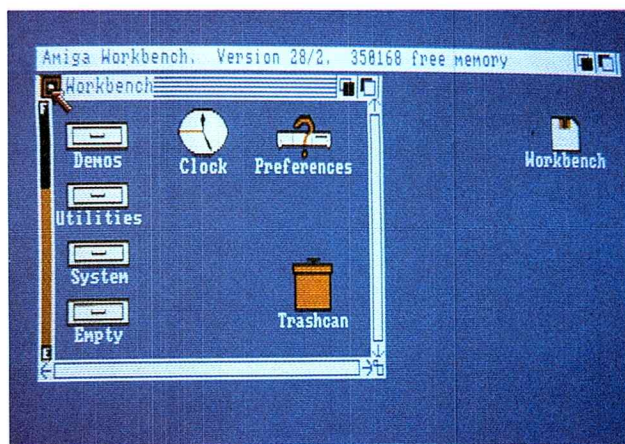
To move what appears in a window, you can drag the Scroll Box:



Selecting the space to either side of the Scroll Box causes the box—and the window—to move in that direction.

Closing Windows

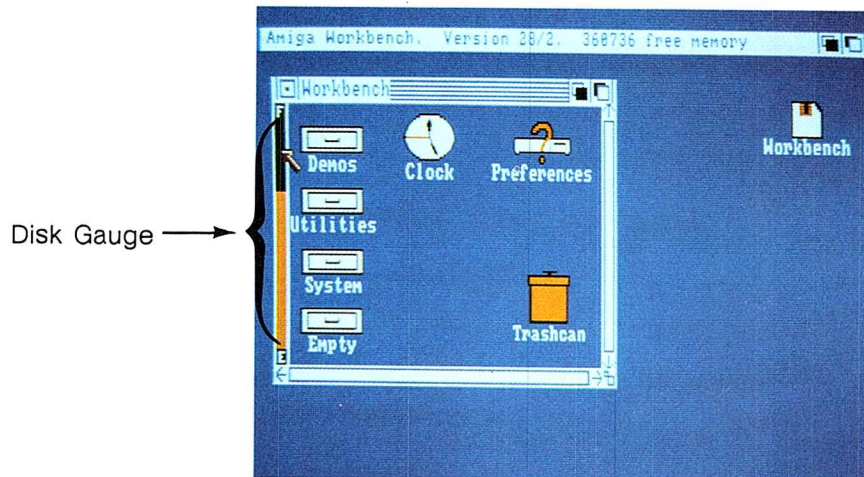
To close a window, select the *Close Gadget*:



You can also close a window for a drawer by selecting its icon, then choosing Close from the Workbench menu.

Disk Gauges

When you open a disk, the window that appears has a *disk gauge* along its left edge:



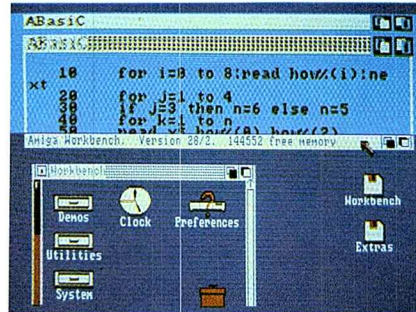
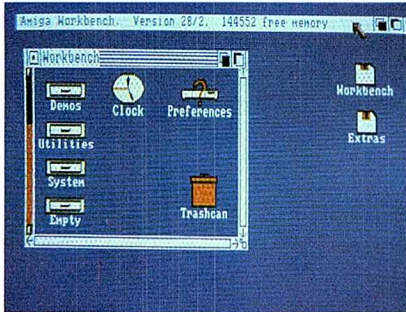
This gauge shows how full the disk is. The closer the colored center bar is to the top, the less free storage space there is on the disk. If the disk is completely full, the colored bar fills the entire space between the "E" and "F" marks.

To free disk space, move tools, projects, or drawers to the Trashcan, then choose Empty Trash from the Disk menu.

Operations Involving Screens

As noted earlier, screens are areas of the display with different video attributes. When a window is opened for a tool, it appears in a screen whose video attributes are appropriate for it.

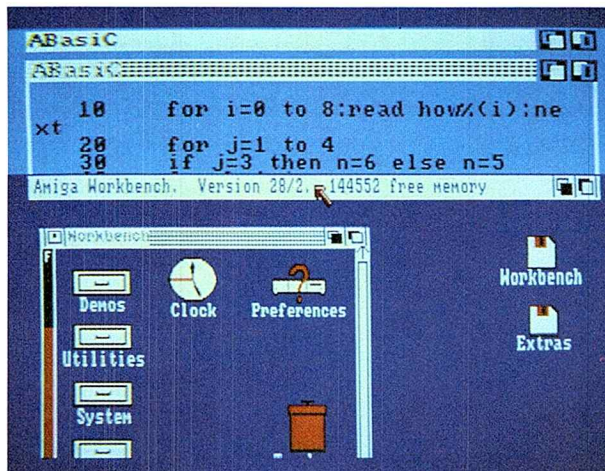
Screens are always as wide as the display, and are no larger than the display. Although the height of a screen is fixed, part of a screen can be off the display:



Screens, like windows, can contain gadgets. Note that a window within the screen can cover the screen's gadgets. If this happens, you must drag or resize the window to reveal the gadgets underneath.

Dragging Screens

To drag a screen, point anywhere in the screen's Title Bar that is not occupied by other gadgets (the screen's *Drag Bar*), hold down the Selection button, then move the mouse:



You can drag a screen down so that part of it is off the bottom of the display. Note that you cannot drag a screen up so that the bottom of the screen is above the bottom of the display.

Moving Screens in Front of Other Screens

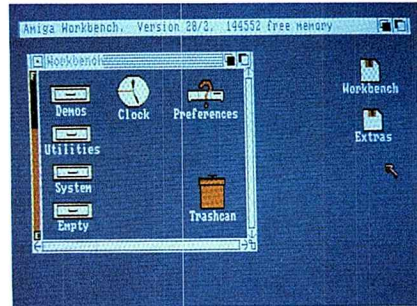
To move a screen in front of other, overlapping screens, select the screen's Front Gadget:



For the Workbench screen, there is a selection shortcut you can use to move it to the front: while holding down the left Amiga key, press the N key.

Pushing Screens Behind Other Screens

To move a screen behind other, overlapping screens, select the screen's Back Gadget:

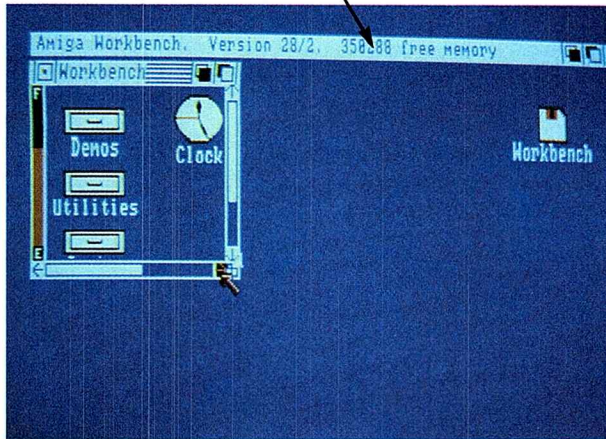


For the Workbench screen, there is a selection shortcut you can use to push it to the back: while holding down the left Amiga key, press the M key.

The Memory Meter

At the top of the Workbench screen is a *memory meter*:

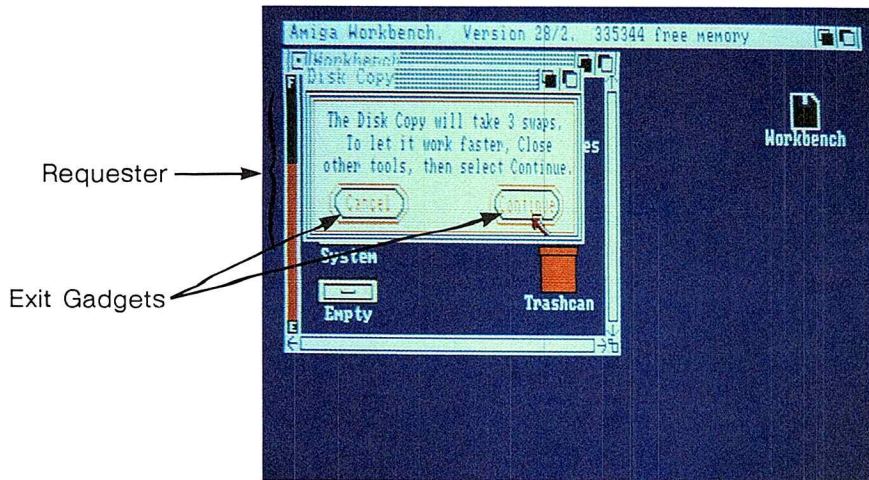
Memory Meter



The meter shows the amount of free *RAM* (*random-access memory*) available to you.

Operations Involving Requesters

A *requester* is an area within a window that a tool uses to communicate with you. Here is an example of a requester:



To respond to a requester, you use the gadget or gadgets it provides. Among the gadgets, there are always one or more *exit gadgets* that you select to close the requester. In many requesters, the "OK" gadget is an exit gadget. Many requesters also have a "Cancel" gadget you select if for any reason you don't want to perform an action.

Alerts are messages the Amiga provides if there is something seriously wrong with your Amiga or with the tools you're using. Alerts are hard to ignore: they appear in boxes with flashing red borders. At the top of alerts are the words "Software Failure" or "Not enough memory."

If you get an alert, jot the number at the bottom of the box on a piece of paper if you can; it will help service people to diagnose the problem.

Operations Involving Disks

Initializing Disks

To use a new disk with the Amiga, it must be *initialized*. If you copy a disk, the new disk is initialized as it receives the copy. To initialize a disk without

making a copy, insert the disk in a disk drive, select the disk icon that appears on the Workbench, then choose Initialize from the Disk menu.

Two warnings:

Initializing a disk destroys any previous information stored on a disk.

Before a disk is completely initialized, the disk drive light will go out, then, after a brief period—from one to ten seconds—it will go on again. Wait for the light to go out a second time before removing the disk. Failure to do so may ruin the disk.

Duplicating Disks

To duplicate a disk, select the icon for the disk, then choose Duplicate from the Workbench menu. Note that when you choose Duplicate, the Amiga makes use of only one disk drive even if there are two or more drives.

Copying Disks

To copy a disk, drag its icon over the icon for a disk that will receive the copy. If you have more than one disk drive, a requester will ask you to insert the disk you want to copy (the *source disk*) into one of the drives, and the disk to receive the copy (the *destination disk*) into another. (Note that “drive 0” referred to in the requester is the internal drive. “Drive 1” is the external drive.)

Two warnings:

Copying a disk destroys any previous information stored on the disk that receives the copy.

If, when you copy a disk, you insert the destination disk in place of the source disk, you will not get a message telling you that you’ve inserted the wrong disk. Be sure to insert the correct disk.

Moving a Tool, Project, or Drawer to a New Disk

To move a copy of a tool, project, or drawer to a new disk, open the disk you want to move it to, then drag the icon into the window for the disk.

Renaming Disks

To rename a disk, select the icon for the disk, then choose Rename from the Workbench menu. A message then appears asking you for a new name. Select the window that appears, type in a name, then press the RETURN key.

Resetting the Workbench

Resetting the Workbench means to set it up again. When you do, you start again with only the Workbench; the Amiga's memory is cleared. If a tool malfunctions, you may be forced to reset before you can resume work. To reset the Workbench, hold down the CTRL key and both Amiga keys at the same time for at least half a second, then release the keys.

WARNING: Always make sure the disk drive lights are off before resetting the Workbench.

Other Workbench Operations

There are four other tasks you perform on the Workbench. You choose each task—straightening up the Workbench icons, displaying the last error message, *redrawing* the display, and saving the positions of icons and windows—from the Special menu for the Workbench.

Cleanup

If a drawer is open and the icon you selected to open the drawer is currently selected, choosing Cleanup straightens up icons in the drawer.

Last Error

Choosing Last Error from the Special menu displays the last message that appeared in the Title Bar for the Workbench. Messages that appear in the Title Bar normally disappear as soon as you select something on the Workbench. (For explanations of error numbers that appear in the Title Bar, see Appendix B, "AmigaDOS Messages.")

Redraw

Choosing Redraw redraws what appears on the screen. Should a tool malfunction, it may affect what appears in a screen. Choosing Redraw from the Special menu restores what appears on the Workbench screen if it has been disturbed.

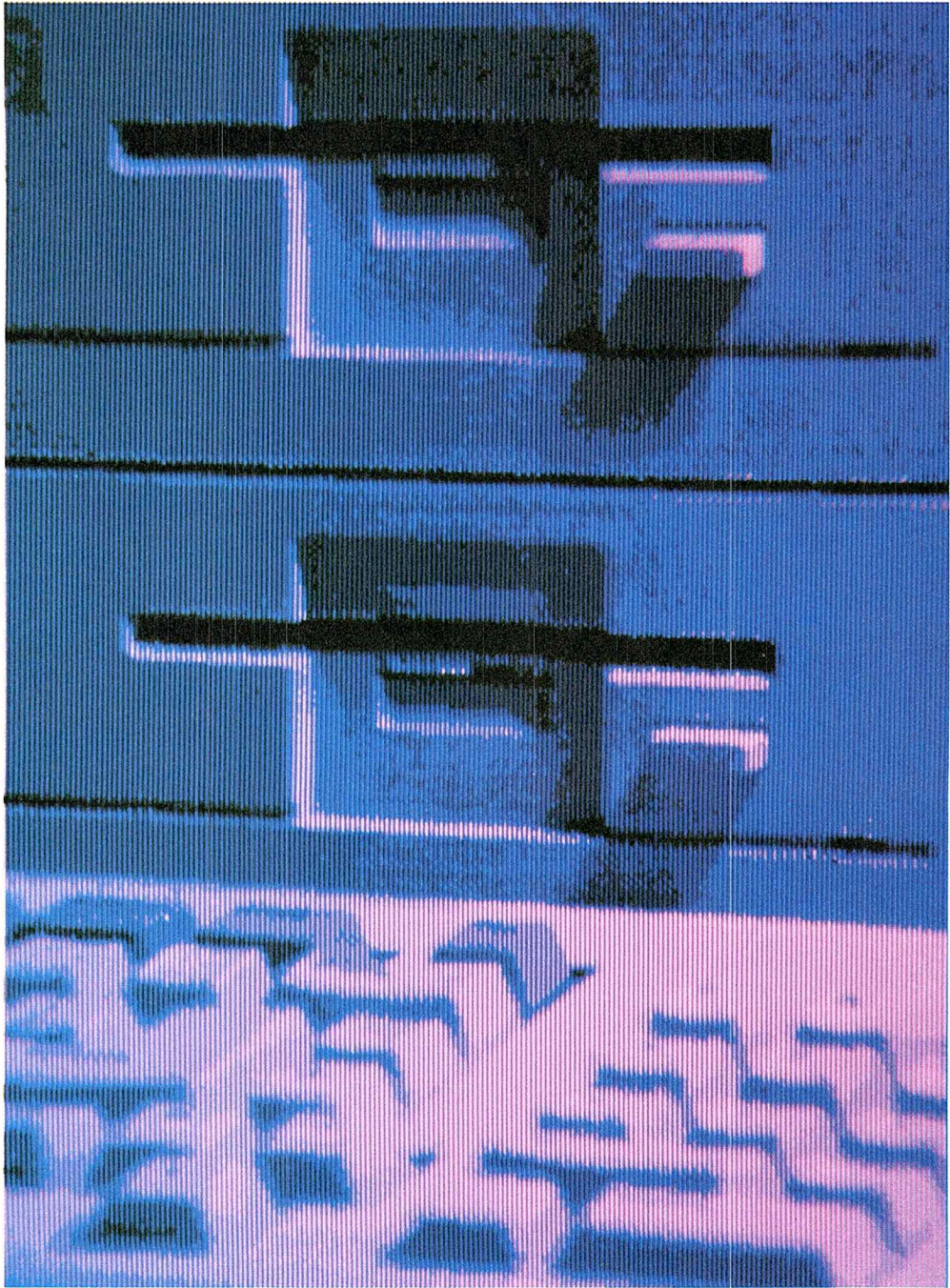
Snapshot

Choosing Snapshot saves on disk the positions of currently selected icons. It also saves the sizes and the positions of windows that appear when you open any of the disk or drawer icons that are selected. (Note, however, that the positions of unselected icons within those windows are not saved.)

Note that you can take a snapshot of more than one icon at a time by using Extended Selection.

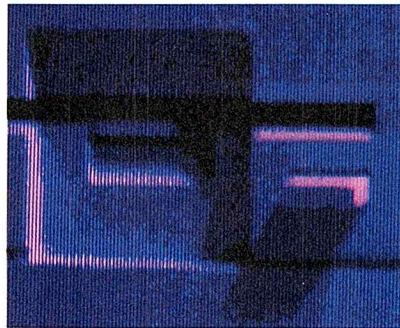
Workbench Tools

For information about Preferences, the tool you use to change many of the Amiga's settings, see Chapter 7. To learn about the *Clock* and the *Notepad*, see Appendix A, "Workbench Tools."



Chapter 5

Adding to the Amiga



There are many ways you can add to your Amiga. You can make it more powerful by adding memory or an extra disk drive. Tools for business and entertainment let you use your Amiga in new and exciting ways. To print your projects, you can choose from several printers, including color printers.

In this chapter, you'll get a quick look at some currently available add-ons. Complete instructions for installing and using these add-ons are included with the add-ons. For more information, and for many add-ons not described here, see your Amiga dealer.

Precautions for Add-Ons

When attaching any add-on, **use only a cable that is specifically designed for the Amiga.** Using a cable that is not properly wired for the Amiga **may damage the add-on.** You can obtain cables designed for the Amiga from your Amiga dealer. If you wish to adapt other cables for use with the Amiga, see Chapter 7 for information about the proper connections.

Before you attach a cable to any of the connectors on the back of the main unit, turn off the Amiga. Attaching a cable when the Amiga is turned on may reset the Amiga. (This precaution does not apply to the connectors labeled "1" and "2" on the right side of the main unit; you can switch add-ons you attach to these connectors at any time.)

When using cables to attach any add-ons, including printers, be sure that the cables are shielded. Using unshielded cables can cause interference to radio and television reception. See Chapter 7 for more information about how to prevent and correct interference.

Adding Memory to the Amiga

With the *Amiga Memory Expansion Cartridge*, you can easily add an additional 256K of random-access memory to your Amiga. The cartridge slides into the front of the Amiga and takes only seconds to install. With the additional memory, you can:

- open additional tools and switch quickly between them.
- use tools that take advantage of extra memory. Many tools work faster when there is more memory available.
- use tools that require more than 256K of memory.

Adding a Disk Drive to the Amiga

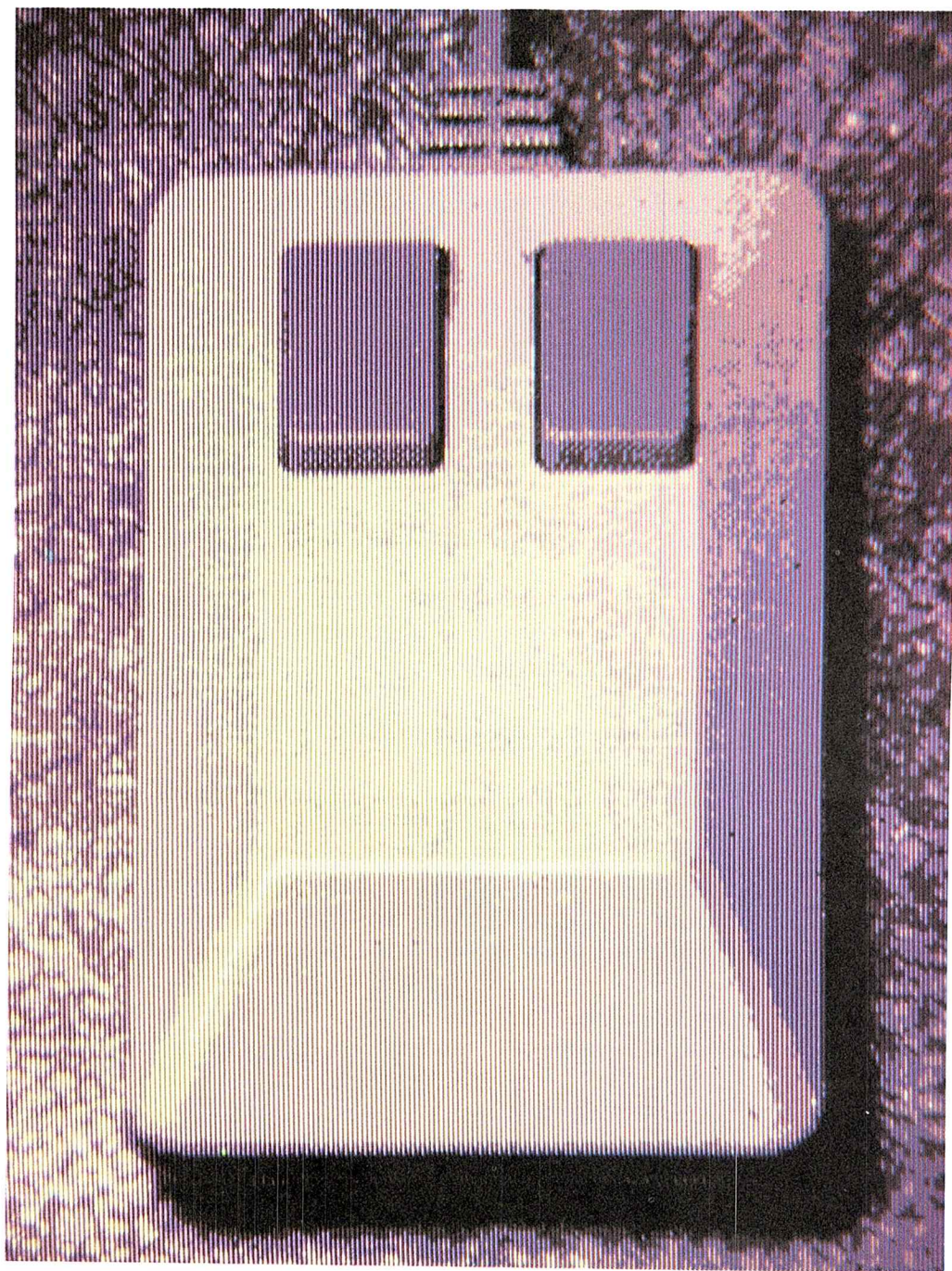
The *Amiga External 3.5 Disk Drive* is identical in storage capacity and performance to the disk drive built into the Amiga. To attach this drive, you simply plug it into the external disk connector on the back of the Amiga. A second disk drive makes it easier and faster to perform many operations, such as copying disks.

Printers for the Amiga

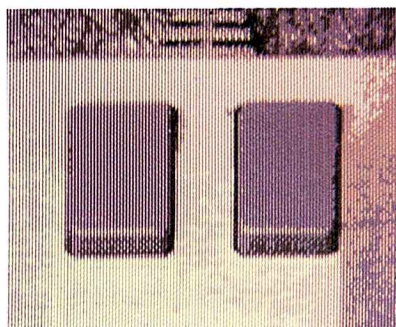
There are five types of printers you can use with an Amiga:

- the Epson® FX-80™ and RX-80™ and the CBM® MPS1000 dot-matrix printers. With these printers, you can produce both text and monochrome graphics.
- the Alphacom® Alphapro 101™, Brother® HR-15XL, Diablo® Advantage D25, Diablo® 630, and Qume® LetterPro 20™ letter-quality printers. These print text identical to that produced by high-quality typewriters. They are, however, slower than most dot-matrix printers and cannot print graphics.
- the Okimate 20™ and Epson® JX-80™ color printers. To use the Okimate 20 with the Amiga, you also need an Okidata “Plug 'n Print” cartridge designed to connect the Okimate 20 to the parallel port of an IBM® PC. The Okimate 20 and the Epson JX-80 can print color images from the Amiga, including paintings you create with Graphicraft™.
- the Diablo® C-150 color printer. This printer uses advanced ink-jet technology to produce high-quality color images.
- Hewlett-Packard LaserJet™ and LaserJet PLUS™ laser printers.

You use the Preferences tool to tell the Amiga which printer you're using and to change a number of settings that affect printers. To learn about Preferences, see Chapter 7.



Caring for the Amiga



Your Amiga needs very little care to keep it working at its best. Observe the precautions in this chapter to keep your Amiga in top shape.

Precautions

Keep the Amiga dry. Keep liquids away from the Amiga as you work. An accidental spill can seriously damage the Amiga.

Keep the Amiga out of direct sunlight. If the case gets too hot, the Amiga won't work reliably. Moreover, temperatures above 140 degrees Fahrenheit (60 degrees Celsius) can damage the Amiga's internal components. Keep it cool.

Keep connectors and the ends of cables clean. Food, especially sticky food, is the worst offender. Any substance that adheres to connectors or the ends of cables can prevent a good electrical connection or, worse, damage the connector.

Keep magnets away from the monitor. Although magnets won't damage the monitor, they can distort the video display. In addition to more obvious magnets, beware of magnets in telephones, loudspeakers, and electric motors. (Note that magnets CAN damage information on disks. Be sure to read "Taking Care of Disks" at the end of this chapter.)

Don't plug anything other than the keyboard into the keyboard connector. Plugging in anything else may damage the Amiga.

Don't put more than 40 pounds (18 kilograms) on top of the main unit. Most monitors weigh less than this, but there are televisions that weigh more.

Don't open the case. If your Amiga needs service, bring it to an Amiga dealer or an approved Amiga Service Center. Opening the case will void the warranty on your Amiga.

Use the mouse on a clean surface. The ball on the bottom of the mouse must be clean to work properly. If the mouse behaves erratically, it may need cleaning. The next section tells how to clean your mouse.

Cleaning the Mouse

To keep the mouse working properly, give it an occasional cleaning. To clean the mouse, you'll need:

- a soft, dry, lint-free cloth
- alcohol or head cleaning fluid for tape recorders
- cotton swabs

Cleaning the mouse takes just a couple of minutes. Here's how you do it:

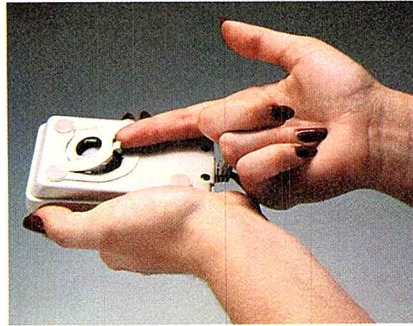


Turn the mouse upside down with its cable toward you. Hold the mouse in both hands and put your thumbs under the two arrows on either side of the ball:

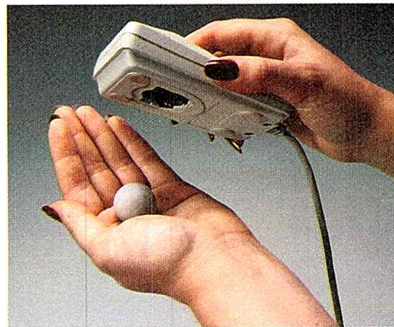




With your thumbs, push firmly in the direction of the arrows to open the cover for the mouse ball. With the mouse upside down, lift off the cover with a fingernail:



Put your hand over the opening, turn the mouse upside down, and catch the ball:



In the opening, you'll see three small metal rollers. Moisten a cotton swab with alcohol or head cleaning fluid and gently swab the surface of each roller. Turn each roller as you swab to clean it all the way around.



With the cloth, wipe off the mouse ball. (Don't use any liquid when cleaning the mouse ball.) When you're done, blow gently into the opening to remove any dust, replace the ball, and slide the cover for the ball back into place.

Taking Care of Disks

To protect the information on your disks, observe these precautions:

Never remove a disk from a disk drive when the disk drive light is on. The disk drive light tells you that the Amiga is using a disk. Taking a disk out too soon may ruin the information on the disk.

Keep disks away from magnets. Microdisks, like audio tapes, store information magnetically. Magnets can ruin the information on a disk. In addition to more obvious magnets, beware of magnets in telephones, loudspeakers, and electric motors.

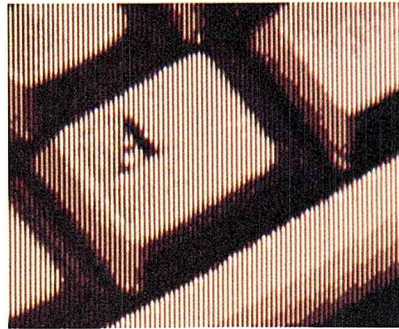
Keep disks dry and away from extreme heat or cold. Microdisks are comfortable at about the same temperatures you are. Don't leave disks in direct sunlight, near heat sources, or in cars parked in the sun.

Don't touch the surface of the disk. A microdisk's metal cover closes automatically whenever you remove the disk from a disk drive. Don't touch the surface of the disk underneath the cover.

Make copies of important disks. The best insurance for the information on a disk is to make a copy of the disk and keep the copy in a safe place. Make a habit of copying an important disk each time you finish working with it.



Reference

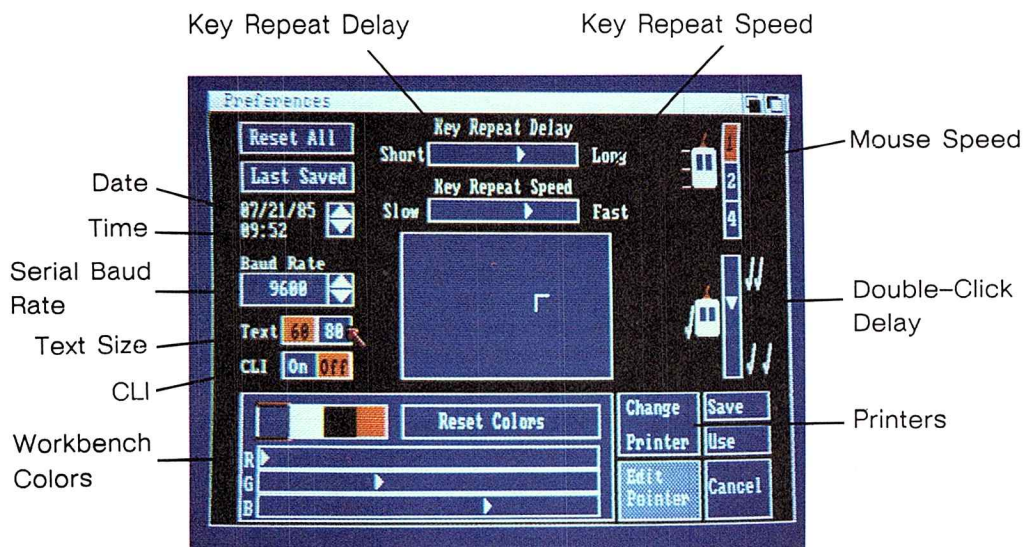


In this chapter, you'll find:

- a complete description of Preferences, the tool you use to change many of the settings of the Amiga
- descriptions of the Amiga input/output connectors
- information about radio and television interference
- specifications for the Amiga

Preferences

Preferences is a tool that lets you see and change many of the settings of your Amiga. These are the settings you can change with Preferences:



Date and Time

To change the date or time, first select the digit you want to change by pointing to it, then clicking the Selection button. With a digit selected, you can:

- select the up arrow to increase the selected digit by one
- select the down arrow to decrease the selected digit by one

The leftmost digits of the date are the number of the month, the middle digits are the day of the month, and the rightmost digits are the last two digits of the year. The time is shown using a 24-hour clock.

Note that if any of the numbers is as large as it can be, increasing it increases the value for the next larger interval of time. For example, if the value for the hours is 11 and the value for the minutes is 59, increasing the digit 9 for the minutes leaves you at 12:00. Conversely, decreasing a value that is as small as it can be decreases the value for the next smaller interval of time.

Key Repeat Speed

To make keys on the keyboard repeat more quickly when you hold down a key, drag the arrow on the slider labeled Key Repeat Speed to the right. To slow down the rate at which keys repeat, drag the arrow to the left.

Key Repeat Delay

When you hold down a key that repeats, there is a delay before the key begins repeating. To increase this delay, drag the arrow on the slider labeled Key Repeat Delay to the right. To decrease the delay, drag the arrow to the left.

Mouse Speed

The three settings for *mouse speed* let you change how far the Pointer moves when you move the mouse. The settings 1, 2, and 4 are the number of inches you move the mouse to move the Pointer roughly a third of the way across the display. The larger the number, the more room you need for the mouse.

Double-Click Delay

You use the *Double-Click Slider* to set the maximum length of time between the two clicks of a double-click. Drag the arrow down to increase the maximum length of time. Drag the arrow up to decrease the maximum time.


Text Size

To make the best use of your monitor, you can choose the size of the characters that appear on the display. Most NTSC monitors and televisions can show 60 characters clearly on each line of the display, while RGB monitors can display 80 characters clearly. If you have an NTSC monitor or television connected to the Amiga, select the gadget labeled 60 to the right of the word Text. If you have an RGB monitor connected to the Amiga, select the gadget labeled 80.

CLI

In addition to the Workbench, the Amiga includes another user interface, the Command Line Interface (CLI). To make an icon for the CLI appear in the System drawer on the Workbench, select the ON gadget immediately to the right of "CLI" on the Preferences screen. (To learn about the CLI, see the *AmigaDOS User's Manual*.)

Display Centering

To center the image on a video display, move the Pointer into the corner of the  symbol that appears in the *Display Centering Gadget*, hold down the Selection button, then move the mouse to change the position of the image.

Baud Rate

If you have an add-on connected to the serial connector of your Amiga, you can change the *baud rate*—the rate at which information is transferred through the serial connector—by selecting the arrows below and to the right of the words Baud Rate. The current baud rate is shown to the left of the arrows. Select the up arrow to increase the baud rate. Select the down arrow to decrease the rate.

Workbench Colors

With Preferences, you can change any of the four colors displayed by the Workbench. Start by selecting the color you want to change from the four colors shown. Below these colors are three sliders labeled R, G, and B. These letters stand for red, green, and blue, the colors that the Amiga combines to create the colors it displays. To modify the color you've selected, you change the amount of red, green, and blue in the color by dragging the arrows along the sliders.

Try dragging the arrows in the sliders and watch how the color changes. With a bit of practice, you'll be able to get the colors you want.

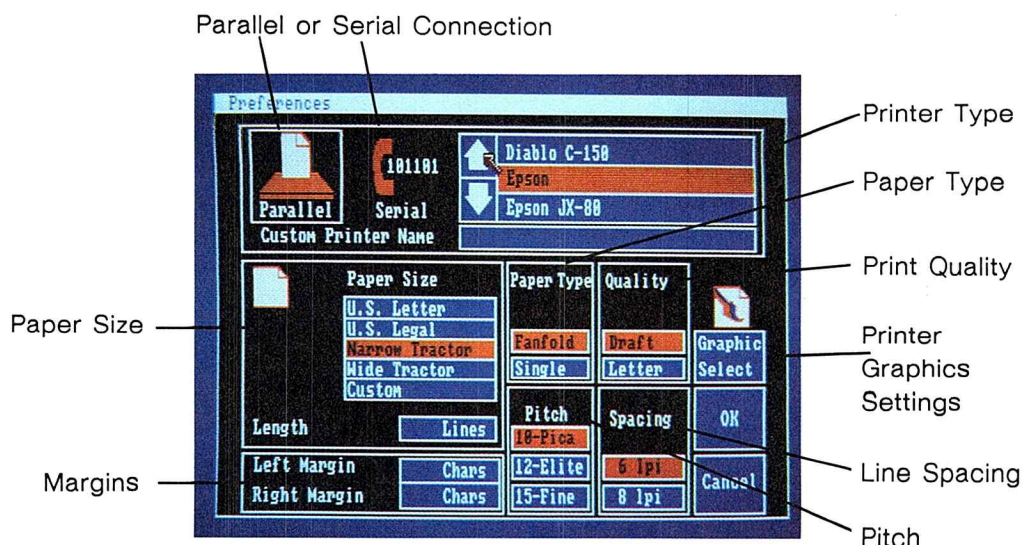
To get back the colors you had before you opened Preferences, select the gadget labeled Reset Colors. (To get back the original Workbench colors—the ones displayed when you inserted the original Workbench disk that came with the Amiga—select the Reset All gadget described below.)

Changing the Pointer

To learn how to change the Pointer with Preferences, see Appendix C, "Changing the Pointer."

Printers

If you've attached a printer to the Amiga, you need to tell the Amiga the type of printer you've attached. You do this by selecting Change Printer. When you do, the *Change Printer Screen* appears:



In this screen, you can select:

- **Printer Type.** The names of printers supported by the Amiga appear in the upper right of the screen. To indicate the printer you're using, select either the up arrow or down arrow until the the name of your printer is highlighted.

Makers of other printers may provide information on disk that allows you to use their printers with the Amiga. If the instructions for your printer state that you are to indicate a project containing this information, select Custom from the list of printers, then select the gadget immediately to the right of the words Custom Printer Name. Type in the name of the project indicated in the instructions, then press the RETURN key on the keyboard.

If you want to attach a printer that is not supported by the Amiga and you do not have a project for it, select Custom from the list of printers, then enter Generic in the Custom Printer Name gadget. For many printers, this will allow you to print plain text, but not graphics or extra type styles such as italics.

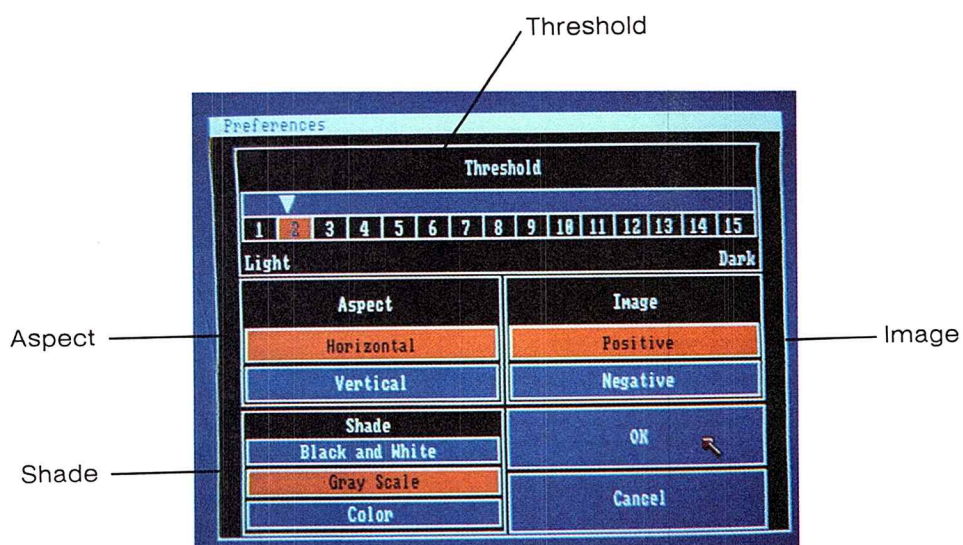
- **Parallel or Serial Connection.** If your printer is attached to the parallel connector on the Amiga, select the gadget labeled Parallel at the upper left of the screen. If it's attached to the serial connector, select the gadget labeled Serial.
- **Paper Size.** There are four preset sizes you can select from: US Letter (8-1/2 inches wide, 11 inches tall), US Legal (8-1/2 inches wide, 14 inches tall), Narrow Tractor (9-1/2 inches wide, 11 inches tall), and Wide Tractor (14-7/8 inches wide, 11 inches tall).

You can use other sizes of paper by selecting Custom. When you select Custom, you must also specify the number of lines that fit on the size of the paper you're using. To do this, select the gadget labeled Length just below the paper sizes, type in the number of lines, then press the RETURN key on the keyboard.

- **Left and Right Margins.** You indicate the width of these margins by specifying numbers of characters from the left-hand edge of the paper. To specify the width of the left margin, select the gadget to the right of the words Left Margin, type in the width, in characters, of the left margin, then press the RETURN key. To specify the width of the right margin, select the gadget to the right of the words Right Margin, type in the width, in characters, for the distance from the left-hand edge of the paper to where the right margin begins, then press the RETURN key.
- **Paper Type.** Select Fanfold if you're using continuous-feed paper. Select Single if you're printing on individual sheets.
- **Quality.** For faster but lower-quality printing, select Draft. For higher-quality printing, select Letter.

- **Pitch.** You use this to select the size of the characters that are printed. You can select from among 10 pitch ("pica"), 12 pitch ("elite"), and 15 pitch ("fine").
- **Spacing.** This lets you select how closely lines are printed on the page. Select either 6 or 8 lines per inch ("lpi").

There is an additional gadget in this screen labeled Graphic Select. Selecting this gadget opens the *Printer Graphics Requester*:



You use this screen to select different ways to print images:

- **Shade** lets you select color printing, gray-scale printing (where colors are represented by different shades of gray), or back-and-white printing (where some colors are printed as pure black, and others as pure white. Whether a color is printed as black or white is determined by the *threshold value* described below.)
- **Aspect** lets you select whether to print normally or “sideways” on the page. Select Horizontal to print “normally,” so that what appears on the top of the display appears along the top edge of the printer paper. Select Vertical to print what appears on the top of the display along the side of the printer paper.
- **Image** lets you print an image as it appears on the display (by selecting Positive) or “reversed” (by selecting Negative). This setting affects only black-and-white and gray scale printing.
- **Threshold**, for black-and-white printing, lets you determine which colors are printed as white, and which as black. You change the Threshold setting by dragging the arrow in the slider below the label Threshold. When the setting for Image is Positive and the Threshold setting is 2, only the darkest color on the display is printed as black, while the rest is white. Increasing the value of the Threshold setting causes more colors to be printed as black. As you increase the setting, the lighter colors are printed as black.

When the setting for Image is Negative, the higher the Threshold setting, the lighter are the colors that are printed as black.

Note that not all these choices apply to all printers. For example, letter-quality printers that use a “daisy wheel” printhead can only produce one quality of printing. To find out what selections apply to your printer, see the documentation provided with the printer.

When you’re done making selections for your printer, select OK to confirm your selections or Cancel to cancel them. Selecting either OK or Cancel returns you to the Preferences window.

Getting Back Preferences

If you'd like to get back the Preferences settings that came with the original Workbench disk, select **Reset All**. If you'd like to get back the last Preferences settings you saved, select **Last Saved**.

Using and Saving Preferences

When you're done with Preferences, select one of the gadgets at the lower right of the window. Select **Save** if you want your settings to take effect now and each time you start up the Workbench with the Workbench disk you're currently using. Selecting **Save** saves your settings on the Workbench disk. Select **Use** if you want your settings to take effect now, but you don't want to save the settings on the Workbench disk for future use. If you change the settings, then decide you don't want them to take effect, select **Cancel**.

Because each Workbench disk keeps its own Preferences settings, different people can save their own settings on separate Workbench disks. To get back your settings, just set up the Workbench using the disk on which you've saved them.

Input/Output Connectors

This section lists pin assignments for several input/output connectors on the Amiga. The information in this section is highly technical and is intended only for those expert in connecting external devices to computers. You do not need this information if you use a cable specifically designed for use with the Amiga and the add-on you want to connect.

For information about connectors not described in this section, see the *Amiga Hardware Manual*.

If you attach add-ons with cables other than those designed for use with the Amiga, note: **some pins on Amiga connectors provide power outputs and non-standard signals. Attempting to use cables not wired specifically for the Amiga may cause damage to the Amiga or to the equipment you connect.** The descriptions below include specific warnings for each connector. For more information about connecting add-ons, consult your Amiga dealer.

In the descriptions that follow, an asterisk (*) at the end of a signal name indicates a signal that is active low.

Serial Connector

In the following table, the second column from the left gives the Amiga pin assignments. The third and fourth columns from the left give pin assignments for other commonly used connections; the information in these two columns is given for comparison only.

WARNING: Pins 14, 21, and 23 on the Amiga serial connector are used for external power. Connect these pins **ONLY** if power from them is required by the external device. The table lists the power provided by each of these pins.

Pin	Amiga	RS232	HAYES®	Description
1	GND	GND		FRAME GROUND
2	TXD	TXD	TXD	TRANSMIT DATA
3	RXD	RXD	RXD	RECEIVE DATA
4	RTS	RTS		REQUEST TO SEND
5	CTS	CTS	CTS	CLEAR TO SEND
6	DSR	DSR	DSR	DATA SET READY
7	GND	GND	GND	SYSTEM GROUND
8	CD	CD	CD	CARRIER DETECT
9				
10				
11				
12		S.SD	SI	
13		S.CTS		
14	-5V	S.TXD		-5 VOLT POWER (50 mA)
15	AUDO	TXC		AUDIO OUT OF AMIGA
16	AUDI	S.RXD		AUDIO INTO AMIGA
17	EB	RXC		BUFFERED PORT CLOCK
18	INT2*			INTERRUPT LINE TO AMIGA
19		S.RTS		
20	DTR	DTR	DTR	DATA TERMINAL READY
21	+5V	SQD		+5 VOLT POWER (100 mA)
22		RI	RI	
23	+12V	SS		+12 VOLT POWER (50 mA)
24	C2*	TXC1		3.58 MHZ CLOCK
25	RESB*			BUFFERED SYSTEM RESET

Parallel Connector

WARNING: Pin 23 on the Amiga parallel connector supplies +5 volts of power. Connect this pin **ONLY** if the power from it is required by the external device. **NEVER** connect this pin to an output of an external device or to a signal ground.

Pins 14–22 are for grounding signals. **DO NOT** connect these pins directly to a shield ground.

Pin	Name	Description
1	DRDY*	DATA READY
2	D0	DATA BIT 0 (Least significant bit)
3	D1	DATA BIT 1
4	D2	DATA BIT 2
5	D3	DATA BIT 3
6	D4	DATA BIT 4
7	D5	DATA BIT 5
8	D6	DATA BIT 6
9	D7	DATA BIT 7
10	ACK*	ACKNOWLEDGE
11	BUSY	BUSY
12	POUT	PAPER OUT
13	SEL	SELECT
14	GND	SIGNAL GROUND
15	GND	SIGNAL GROUND
16	GND	SIGNAL GROUND
17	GND	SIGNAL GROUND
18	GND	SIGNAL GROUND
19	GND	SIGNAL GROUND
20	GND	SIGNAL GROUND
21	GND	SIGNAL GROUND
22	GND	SIGNAL GROUND
23	+5V	+5 VOLTS POWER (100 mA)
24		
25	RESET*	RESET

RGB Monitor Connector

WARNING: Pins 21, 22, and 23 on the RGB monitor connector are used for external power. Connect these pins **ONLY** if power from them is required by the external device. The table lists the power provided by each of these pins.

Pin	Name	Description
1	XCLK*	EXTERNAL CLOCK
2	XCLKEN*	EXTERNAL CLOCK ENABLE
3	RED	ANALOG RED
4	GREEN	ANALOG GREEN
5	BLUE	ANALOG BLUE
6	DI	DIGITAL INTENSITY
7	DB	DIGITAL BLUE
8	DG	DIGITAL GREEN
9	DR	DIGITAL RED
10	CSYNC*	COMPOSITE SYNC
11	HSYNC*	HORIZONTAL SYNC
12	VSYNC*	VERTICAL SYNC
13	GNDRTN	RETURN FOR XCLKEN*
14	ZD*	ZERO DETECT
15	C1*	CLOCK OUT
16	GND	GROUND
17	GND	GROUND
18	GND	GROUND
19	GND	GROUND
20	GND	GROUND
21	-5V	-5 VOLTS POWER (50 mA)
22	+12V	+12 VOLTS POWER (175 mA)
23	+5V	+5 VOLTS POWER (300 mA)

TV Modulator Connector

WARNING: Pin 7 on the TV modulator connector supplies +12 volts of power. Connect this pin ONLY if power from it is required by the external device.

Pin	Name	Description
1		
2	GND	GROUND
3	AUDIO LEFT	LEFT AUDIO CHANNEL
4	COMP VIDEO	COMPOSITE VIDEO OUTPUT
5	GND	GROUND
6		
7	+12V	+12 VOLTS POWER (60 mA)
8	AUDIO RIGHT	RIGHT AUDIO CHANNEL

Mouse/Game Controller Connectors

There are connectors labeled "1" and "2" on the right side of the Amiga. If you use a mouse to control the Workbench, you must attach it to connector 1 (the connector closest to the front of the Amiga). You can attach game controllers to either of the connectors. To use a light pen, you must attach it to connector 1. The following tables describe mouse, game controller, and light pen connections.

WARNING: Pin 7 on each of these connectors supplies +5 volts of power. Connect this pin ONLY if power from it is required by the external device.

Connectors 1 and 2: Mouse Connections

Pin	Name	Description
1	MOUSE V	MOUSE VERTICAL
2	MOUSE H	MOUSE HORIZONTAL
3	MOUSE VQ	VERTICAL QUADRATURE
4	MOUSE HQ	HORIZONTAL QUADRATURE
5	MOUSE BUTTON 2	MOUSE BUTTON 2
6	MOUSE BUTTON 1	MOUSE BUTTON 1
7	+5V	+5 VOLTS POWER (125 mA)
8	GND	GROUND
9	MOUSE BUTTON 3	MOUSE BUTTON 3

Connectors 1 and 2: Game Controller

Pin	Name	Description
1	FORWARD*	CONTROLLER FORWARD
2	BACK*	CONTROLLER BACK
3	LEFT*	CONTROLLER LEFT
4	RIGHT*	CONTROLLER RIGHT
5	POT X	HORIZONTAL POTENTIOMETER
6	FIRE*	CONTROLLER FIRE
7	+5V	+5 VOLTS POWER (125 mA)
8	GND	GROUND
9	POT Y	VERTICAL POTENTIOMETER

Connector 1: Light Pen Connections

Pin	Name	Description
1		
2		
3		
4		
5	LIGHT PEN PRESS	LIGHT PEN TOUCHED TO SCREEN
6	LIGHT PEN*	CAPTURE BEAM POSITION
7	+5V	+5 VOLTS POWER (125 mA)
8	GND	GROUND
9		

Radio and Television Interference

Your Amiga generates and uses radio frequency energy. If it not installed and used properly, that is, in strict accordance with the instructions in this manual, it may cause interference to radio and television reception. The Amiga has been certified to comply with the limits for a Class B computing device, pursuant to subpart J of Part 15 of the Federal Communication Commission's rules, which are designed to provide reasonable protection against radio and television interference in a residential installation. If you suspect interference, you can test the Amiga by turning it off and on. If the Amiga does cause interference, try the following:

- Reorient the antenna or AC plug on the radio or television.
- Change the relative positions of the Amiga and the radio or television.
- Move the Amiga farther away from the radio or television.
- Plug either the Amiga or the radio or television into a different outlet so that the Amiga and the radio or television are on different circuits.

Use only shield-grounded cables when connecting peripherals (computer input-output devices, terminals, printers, etc.) to the Amiga. All peripherals must be certified to comply with Class B limits. Operation with non-certified peripherals is likely to result in interference to radio and television reception.

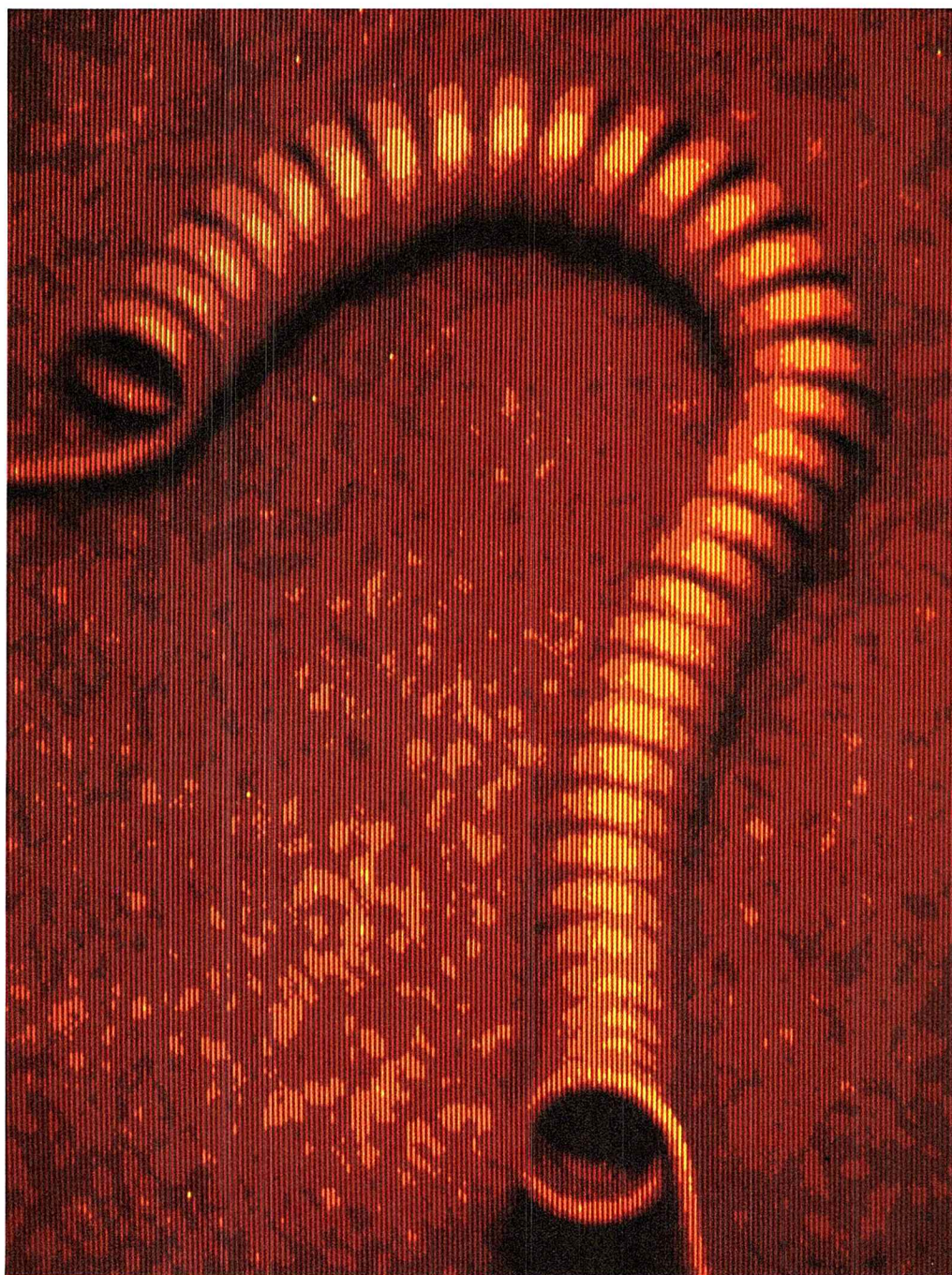
Your house AC wall receptacle must be a three-pronged type (AC ground). If not, contact an electrician to install the proper receptacle. If a multi-connector box is used to connect the computer and peripherals to AC, the ground must be common to all units.

If necessary, consult your Amiga dealer or an experienced radio-television technician for additional suggestions. You may find the following FCC booklet helpful: "How to Identify and Resolve Radio-TV Interference Problems." The booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, stock no. 004-000-00345-4.

Amiga Specifications

Central Processor	Motorola MC68000
Memory	256K bytes RAM expandable to 512K
Disks	3-1/2 inch double-sided microdisks with 880K bytes formatted storage capacity per disk
Mouse	Mechanical, .13 mm/count (200 counts per inch)

Interfaces	RS-232 serial interface
	Centronics®-compatible parallel interface
	External disk interface
	Mouse/Game controller interface
	Additional game controller interface
	Keyboard interface
	Two audio outputs for stereo sound
	Memory cartridge interface
	Expansion interface
Supported Monitors	RGB, NTSC (composite video), and standard televisions
Power Requirements	99 to 121 volts AC 54 to 66 Hz
Temperature Requirements	For operation: 5 to 45 degrees Celsius (41 to 113 degrees Fahrenheit)
	For storage: -40 to 60 degrees Celsius (-40 to 140 degrees Fahrenheit)
Humidity Requirements	20% to 80% relative humidity, non-condensing
Maximum Weight the Main Unit Can Support	40 pounds (18 kilograms)



Appendices



Appendix A: Workbench Tools

The Workbench disk contains several tools:

- Preferences, the tool you use to change many of the settings of the Amiga. Preferences is described in Chapter 7.
- Demonstrations that show the graphics capabilities of the Amiga. You can find these tools in the *Demos drawer* on the Workbench. To start a demonstration, select one of the icons in the Demos drawer, then choose Open from the Workbench menu. To stop a demo, select the Close Gadget in the upper left-hand corner of the demonstration's window.
- The *Clock* and the *Notepad*. These tools are described below.

The Clock

The Clock tool lets you show the current time. In addition, you can use the Clock as an alarm clock.

Setting the Time

To set the time for the Clock, use the Preferences tool. (To learn about Preferences, see Chapter 7.)

Opening the Clock

To open the Clock, select the Clock icon in the Workbench disk drawer, then choose Open from the Workbench menu. When you do, an analog clock with a second hand appears in a window.

Using the Clock Menus

Note that to choose from the menus described below, you must first select the window in which the Clock appears.

Changing the Clock from Analog to Digital

Choose the type of clock you want (either analog or digital) from the Type menu. The type currently chosen is indicated by a check mark.

Changing the Size and Position of the Clock

To change the size of an analog clock, drag the Sizing Gadget at the lower right-hand corner of the window. (You cannot change the size of a digital clock's window.) To move the Clock to a new location, drag the window by its Drag Bar.

Changing from a 12- to a 24-Hour Clock

You can choose either the 12 Hour or 24 Hour setting from the Mode menu. The current setting is indicated by a check mark.

Displaying the Seconds

If you don't want to display the second hand on an analog clock or the digits for the seconds on a digital clock, select the Seconds gadget in the lower left corner of the Clock window. Selecting the Seconds gadget again restores the second hand or the seconds digits.

Setting the Alarm

The items in the Alarm menu let you use the Clock as an alarm clock. To set the alarm, choose Set. In the requester that appears, the time is shown using either a 24-hour clock or a 12-hour clock with "AM" or "PM" indicated. To change the hour setting, point to the digits for the hours, click the Selection button, then select either the up arrow (to move the time ahead) or the down arrow (to move the time back). To change the setting for the minutes, point to the digits for the minutes, click the Selection button, then select either the up or down arrow. Selecting AM or PM switches the setting. When the time is set correctly, select USE. If, instead, you want to restore the previous alarm setting, select CANCEL.

To turn on the alarm clock, choose Alarm On from the Alarm menu. To turn it off, choose Alarm Off.

The “alarm” is a brief flash on the display (the same flash that appears when an error occurs) accompanied by an equally brief “beep” sound if your Amiga is attached to audio equipment.

Closing the Clock

To close the Clock, select the Close gadget in the upper right-hand corner of the window.

The Notepad

With the Notepad tool, you can keep notes or create short documents. You can find the Notepad in the Utilities drawer on the Workbench disk.

Opening the Notepad

You open the Notepad by selecting its icon, then choosing Open from the Workbench menu. When you do, a window for the Notepad appears.

Entering Text

To enter text, select the Notepad window if it isn't already selected, then type. When you type, the characters you type appear to the left of the Text Cursor (the vertical bar that appears in the window). As you add characters, any characters to the right of the Text Cursor move to the right or, if they're at the right edge of the window, down a line. (To see how this works, try adding characters.)

When you're typing and you reach the bottom of the window, the contents of the window are scrolled upward.

Moving the Text Cursor

To move the Text Cursor, point to a place within your note, then click the Selection button. (Note that you cannot move the Pointer to a point in the window beyond where you've entered characters.) You can also move the Text Cursor by pressing the cursor keys. Note that when you press the up or down cursor key and you reach the top or bottom of the window, the contents of the window are scrolled.

Changing the Size of the Notepad Window

You can change the size of the Notepad window by dragging the Sizing Gadget at the lower right. When you do, your note is automatically reformatted.

Moving from Page to Page

There are two additional gadgets in the Notepad window. The gadget at the lower left of the window is the Next Page Gadget. Select this gadget to display the next page of your note. The gadget at the upper right is the Previous Page Gadget. Select this gadget to display the previous page of your note.

The Notepad Menus

The Notepad has four menus: Project, Font, Style, and Format. These are described below.

The Project Menu

New

Choose New to start a new note.

Open

Choose Open to open a note you previously saved. When you do, a requester appears. Select the gadget to the left of "Name:", then change the name, if one appears, to make it the name you want.

To change what appears in the gadget, press the DEL key to delete the characters at and to the right of the Text Cursor. Press the BACKSPACE key to delete characters to the left of the cursor. You can use the left and right cursor keys to move the cursor. You can erase what you've typed in the gadget by pressing the right Amiga key and the X key at the same time. You can get back what was in the gadget before you started by pressing the right Amiga key and the Q key at the same time.

When you're done, press the RETURN key, then select the OK Gadget. The note whose name you type replaces the current note.

Save

Choose Save to save the current note. If you haven't already saved your note, a requester appears and lets you give it a name. Select the gadget to the left of "Name:", type in a name, press the RETURN key, then select the OK Gadget.

Save As

Choose Save As to save the current note under a new name. When you do, a requester appears. Select the gadget to the left of "Name:", then change the name, if one appears, to make it the name you want.

To change what appears in the gadget, press the DEL key to delete the characters at and to the right of the Text Cursor. Press the BACKSPACE key to delete characters to the left of the cursor. You can use the left and right cursor keys to move the cursor. You can erase what you've typed in the gadget by pressing the right Amiga key and the X key at the same time.

You can get back what was in the gadget before you started by pressing the right Amiga key and the Q key at the same time.

When you're done, press the RETURN key, then select the OK gadget.

Note that when choosing either Save or Save As, your note is saved in the drawer whose window was selected when you opened the Notepad.

Print

To print your note, you choose one of the items from the Print submenu. (To choose from the submenu, point to Print, then, with the Menu button still held down, move the Pointer to the right, point to one of the options—explained below—then release the Menu button.)

Choosing the Auto-size option prints an image that is approximately the same size as the image on the display. By choosing the Small option, you print an image whose width is one-quarter the width of the printer paper. (You use the Preferences tool to specify the width of the paper you're printing on. However, note that specifying different dimensions for the paper affects only notes printed when Draft option is chosen. The Draft option is described below.) By choosing the Medium option, you print an image whose width is one-half the width of the printer paper. By choosing the Large option, you print an image whose width is the full width of the printer paper.

If the Graphic option in the Print As submenu is chosen (see below), you print, for each page, a picture of the Notepad window with the page within it. If the Draft option in the Print As submenu is chosen, you print only the text of the note.

Print As

From the submenu, choose Graphic if you want to print a pixel-by-pixel representation of the window in which your note appears. (If you have a color printer, you can print the note in color.) Choose Draft if you want to print only the text of the note.

Quit

Choose Quit when you're done and want to close the Notepad.

The Font Menu

From the Font menu, you can choose the typeface and type size for your note. The names of the seven different typefaces (Topaz, Ruby, Diamond, Opal, Emerald, Garnet, and Sapphire) are shown when you open the menu, while the available type sizes for each typeface are shown in submenus. To choose from the Font menu, point to a name of a typeface, then, with the Menu button still held down, move the Pointer to the right, point to a type size, then release the Menu button.

To see the available typefaces and type sizes, type in a note, then try each of the choices. The currently chosen typeface is shown with a check mark to the left of the menu item. In addition, the currently chosen type size is shown with a check mark if there is more than one size for the currently chosen typeface.

Note that the currently chosen typeface and type size apply to the entire note.

When you choose a new typeface and type size, a previous type size you chose for a different typeface may still have a check mark to the left of it. If this is the case and you attempt to choose the previous typeface again, nothing may happen. If this should occur, choose another type size for the previous face, then choose the size you want.

The Style Menu

From this menu, you can choose either standard (Plain) characters for your note, or change the type style by choosing Italic, Bold, Underlined, or any combination of the three. At any point in your note you can choose a new type style; this sets a marker in your note and all the characters from this marker to the next (or to the end of the file, if there are no other markers)

are changed to the new style. Try the different choices and watch how your note changes.

Note that when you choose *Italic*, **Bold**, or Underline, your choice stays in effect until you choose **Plain**.

There are command-key shortcuts for each of the items in this menu: press the right Amiga key and the P key at the same time to choose **Plain**; the right Amiga key and the I key to choose *Italic*; the right Amiga key and the B key to choose **Bold**; the right Amiga key and the U key to choose Underline. As a reminder, the shortcut for each item is shown in the menu to the right of the item.

The Format Menu

Paper Color

Choose **Paper Color** to change the background color for your note. You then choose a color from the four shown in the submenu (see the description of the **Font** menu above to learn how to choose from a submenu). The currently chosen color is indicated by a check mark.

Pen Color

Choose **Pen Color** to change the color of the characters in your note. You then choose a color from the four shown in the submenu (see the description of the **Font** menu above to learn how to choose from a submenu). The currently chosen color is indicated by a check mark.

Be sure that the pen color is different from the paper color; if you don't, you won't be able to read your note.

Appendix B: AmigaDOS Messages

When error messages appear, they often include *error numbers*. These numbers are generated by *AmigaDOS*, the Amiga disk operating system. This appendix lists error numbers that are likely to appear when you're using the Workbench, together with the AmigaDOS message that corresponds to the number and suggestions for what to do.

For more information about the meaning of a specific error number and for explanations of error numbers not listed here, see the *AmigaDOS User's Manual*.

- 103 Out. Of Memory: The Amiga needs more memory to perform an operation. Close one or more windows, then try the operation again. In rare cases, you may have to reset the Amiga to reclaim sufficient memory to continue.

This error can also occur if you attempt to open a tool that requires more memory than is installed in your Amiga. For information about how much memory a tool needs, see the manual for the tool.

- 121 Not an Object Module: This error may occur if you try to open a tool that has been damaged. Try making a new copy of the tool from your original disk.
- 202 Object in Use: Another tool is using the project or tool you want. You may have to wait for the other tool finish.
- 203 Object Exists: You cannot give an object the same name as an existing object. Either give the object a different name or delete the other object so you can reuse its name.
- 205 Object Not Found: A tool or project that the Amiga needs to locate is not present on the disk. This can happen if you've (1) moved a tool to a different drawer, then try to open a project created with that tool, or (2) renamed either the System or Utilities drawer on the Workbench disk, then tried to open a project created with a tool that's in the drawer, or (3) the disk containing a tool you need is not in a disk drive.

- 210 Invalid Stream Name: When renaming a tool, project, drawer, or disk, you have used an invalid character.
- 213 Disk Not Validated: If you've either removed a disk or reset the Workbench while the disk drive light was on, the information on that disk may be unusable. This error may also occur if the disk surface has been damaged. There is currently no way to recover information from a disk that cannot be validated.
- 214 Disk Write Protected: You have asked the Amiga to add information to your disk when the protect tab is in the protected position. (See pages 3-2 and 3-3 for information about protect tabs.) Either change the position of the protect tab or use a disk whose protect tab is in the unprotected position.
- 216 Directory Not Empty: You have tried to delete a drawer that still contains tools, projects, or other drawers. You must empty a drawer before deleting it.
- 218 Device Not Mounted: This error occurs when a tool needs a disk that is not currently in a disk drive.
- 221 Disk Full: There is not enough free storage on the disk to do what you have requested. Try emptying the Trashcan (see page 4-17). If this doesn't work, you must either delete objects on the disk or use a different disk.
- 222 File Delete Protected: A tool, project, or drawer cannot be deleted because it is protected. Select the icon for the object, choose Info from the Workbench menu, then change its status to DELETABLE.
- 225 Not a DOS Disk: Either the disk has not been formatted or it is a Kickstart disk. Use a different disk, or, if you wish to erase all information currently on the disk, choose Initialize from the Disk menu.
- 226 No Disk in Drive.

Appendix C: Changing the Pointer

You can use the Preferences tool to modify the Pointer. The following instructions tell you how.

Opening Preferences

You can find the Preferences tool on the Workbench disk. To open Preferences, insert the Workbench disk, select the icon for the Workbench disk when it appears, then choose Open from the Workbench menu. When the Preferences icon appears, select it and choose Open from the Workbench menu.

To learn more about Preferences, see Chapter 7.

The Pointer Editing Window

When the Preferences screen appears, select the Edit Pointer Gadget that appears near the lower right-hand corner. In a moment, a window appears. In this window—the *Pointer Editing Window*—a magnified image of the Pointer appears at the upper left. It is this magnified image that you modify to change the Pointer. To the right of the magnified view are copies of the Pointer that appear against each of the four Workbench colors; these copies let you judge how the Pointer will look against the colors on the Workbench.

Changing the Colors

The colors you use to draw the Pointer appear near the bottom of the window. Note that these colors can be different from those used for the Workbench. You can modify the three colors to the left in the same way you modify the Workbench colors: by changing the R, G, and B values for each. The rightmost “color” is not a color at all: any parts of the Pointer you draw with it are transparent. When you move the Pointer, colors on

the Workbench behind any transparent parts of the Pointer show through. If, after you modify the Pointer colors, you want get back the last colors that were saved, select the ResetColor Gadget.

Changing the Pointer

To modify the Pointer, select one of the colors or transparent, point to a place in the magnified view where you want a pixel of that color, then click the Selection button. If you want to start from scratch, select Clear to make all the pixels transparent. (If, after you make changes, you'd rather have the old Pointer back, select Restore.)

Changing the Point

Every Pointer has a single pixel called the *point*. To point to something on the display, you position the Pointer so that this pixel is over it.

In the magnified view of the Pointer, the point is indicated by a smaller square within one of the pixels. To change the Pointer's point, select Set Point, point to the pixel in the magnified view you want as the point, then click the Selection button.

When you're done and are happy with the Pointer you've created, select OK. To get back to the main Preferences screen without changing the previous Pointer, select Cancel.

Glossary

add-on	A printer, game controller, modem, or other external component you use with an Amiga.
alert	A message displayed when there is a serious problem with an Amiga.
ALT key	One of two keys next to the Amiga keys at the bottom of the keyboard .
Amiga keyboard	The keyboard similar to a typewriter's attached to an Amiga.
Amiga key	One of the two keys on an Amiga keyboard to the left and right of the Space Bar . You use the left Amiga key for selection shortcuts and the right Amiga key for menu shortcuts . You also use the Amiga keys when operating the Amiga without a mouse .

Amiga Memory Expansion Cartridge	A cartridge you plug into the front of the Amiga to add 256K of memory .
Amiga Monitor	An RGB monitor made for use with the Amiga.
AmigaDOS	The Amiga disk operating system.
audio connector	The connector you use when attaching audio equipment with an Amiga.
audio signal	The output from one of the two audio connectors on the Amiga.
available menu item	An item in a menu that you can choose .
Back Gadget	A gadget you select to move a window or screen behind other windows or screens that overlap it.
baud rate	The rate at which information is transferred through the serial connector .
cable	A set of insulated wires used either to connect the parts of the Amiga or to connect add-ons to the Amiga.
choose	To pick a menu item . You normally choose menu items with the aid of the Menu button .
chosen option	An option that is currently in effect.
click	1. To press and release a mouse button . 2. The action you perform when you click .
Clipboard	A place where parts of a project that you cut or copy are kept.
clipping	A part of a project that has been cut or copied and put on the Clipboard .
Clock	A tool that lets you display the time on the Workbench .

close	1. To remove a window , requester , or screen from the display . 2. To put away a tool or project .
Close Gadget	A gadget that you select to close a window or screen .
Color Palette	The set of colors available in a screen .
column	A set of adjoining pixels or characters that form a vertical line on the video display .
command	A menu item that, when you choose it, instructs the Amiga to perform a task. <i>Compare</i> option .
composite video monitor	<i>See</i> NTSC monitor
connector	Any of the places on the outside of the Amiga which you use to attach external equipment.
copy	To replicate a tool , project , drawer , or disk .
cursor key	One of four keys with an arrow on top at the right of the keyboard . You press these keys either to move the Text Cursor or, by pressing an Amiga key at the same time, to move the Pointer .
custom screen	A screen created by a tool for its own use and, optionally, for use by other tools.
cut	To remove part of a project and place it on the Clipboard . <i>Compare</i> erase .
Demos drawer	A drawer on the Workbench disk in which demonstration tools are kept.
destination disk	When copying disks , the disk that receives the copy. <i>Compare</i> source disk .
discard	To dispose of a project , tool , or drawer by putting it in the Trashcan .
disk	A medium for storing and retrieving information.
disk drawer	A drawer that contains the contents of a disk .

disk drive	A device for reading information from and saving information on a disk .
disk drive light	A light on the front of a disk drive that shows when the disk cannot safely be removed.
disk gauge	A indicator at the left of the window for a open disk that shows how much free storage is available.
display	That which appears on a video monitor or television.
Display Centering Gadget	A gadget provided by Preferences for centering the image on the display .
double click [n.]	The action you perform when you quickly press and release a mouse button twice.
double-click [v.]	To quickly press and release a mouse button twice.
Double-Click Slider	A gadget provided by Preferences for changing the maximum length of time between the two clicks of a double click .
drag	To move an icon, gadget , window , or screen by putting the Pointer over what you want to move, holding down the Selection button , and moving the mouse.
Drag Bar	That portion of a Title Bar that contains no gadgets . You drag the Drag Bar to move a window or screen.
drawer	A place where tools , projects , and other drawers are kept.
edit	To change the contents of a project .
empty	To remove from the Trashcan any projects , tools , or drawers you've discarded . When you empty the Trashcan , you can no longer get back any of the projects , tools , or drawers that were in it.

erase	To remove part of a project without putting what you've removed on the Clipboard . <i>Compare cut.</i>
error numbers	Numbers that identify AmigaDOS errors.
exit gadget	A gadget in a requester that you select to close the requester .
Extended Selection	A technique for selecting more than one icon or gadget at a time. To use it, you select with the Shift key held down.
Extras disk	One of three microdisks packaged with the Amiga.
feature	A noteworthy property of a tool .
Front Gadget	A gadget that you select to move a window or screen to the front of other windows or screens that overlap it.
gadget	Any of the facilities provided within a window , requester , or screen , such as Scroll Bars , Sizing Gadgets , and Close Gadgets , that you use to change what's being displayed or to communicate with a tool .
ghost [adj.]	Displayed less distinctly to indicate unavailability.
ghost gadget	An gadget that is displayed less distinctly to indicate that it is not currently available.
ghost icon	An icon that is displayed less distinctly to indicate that it is not currently available.
ghost menu item	A menu item that is displayed less distinctly to indicate that it is not currently available.
Graphicraft	The Amiga graphic arts tool .
highlight	To display something in a way that distinguishes it. Normally, something is highlighted to indicate that it is selected .
hold down	To press a mouse button or a key on the keyboard without releasing it.

icon	A visual representation of a tool , project , drawer , or disk .
initialize	To prepare a disk so that it can be used by an Amiga.
item = menu item	
key	Any of the switches on a keyboard .
Key Repeat Slider	A gadget provided by Preferences for changing the speed at which keys on the keyboard repeat when you hold them down.
keyboard	A set of keys used for typing or for giving other information to an Amiga.
keyboard cable	The cable used to connect the keyboard to the main unit .
keyboard connector	The connector on the main unit to which you attach the keyboard cable .
Kickstart disk	A microdisk that contains information an Amiga needs to begin operating.
Look Again	A gadget in Open Requesters that you select to update the Project List .
main unit	The largest component packaged with the Amiga. The main unit contains the central processor and other circuitry, memory , and an internal disk drive .
memory	Electronic circuits used to store information.
memory meter	The indicator in the Title Bar for the Workbench screen that shows the amount of free RAM in bytes.
menu	A list of items you can choose from.
Menu Bar	A strip at the top of a screen that contains menu titles . The menu bar for the selected window appears when you hold down the Menu button .

Menu button	The right-hand button on the mouse .
menu item	One of the choices in a menu .
menu shortcut	A way of choosing a menu item by pressing a key on the keyboard while holding down the right Amiga key .
menu title	The name that for a menu that appears in the Menu Bar .
microdisk	A 3 1/2-inch flexible disk .
mouse	A device you move on a flat surface to move the Pointer .
mouse ball	The ball on the bottom of the mouse that rolls as you move the mouse.
mouse button	One of the two buttons on a mouse .
mouse speed	A option provided by Preferences for varying how many inches you must move the mouse to move the Pointer roughly a third of the way across the display .
Multiple Choice	A technique for choosing more than one option at a time. To use it, you hold down the Menu button , then click the Selection button with the Pointer over the options you want.
Notepad	A tool provided with the Workbench for writing short messages.
NTSC monitor	A type of color monitor that can be used with the Amiga.
OK Gadget	A gadget in a requester that you select to carry out what you've asked for in the requester.
object	A tool , project , drawer , or disk .
open	<ol style="list-style-type: none"> 1. To display a window, requester, or screen. 2. To make a tool or project available.

Open Requester	A requester from which you select a project you want to open.
option	A feature of a tool that, once you choose it, persists until you choose another, mutually exclusive feature.
palette = color palette	
parallel port	A connector on the back of the Amiga that you use to attach printers and other add-ons.
paste	To copy the contents of the Clipboard into a project.
peripheral = add-on	
pixel	One of the small elements that together make up the video display.
pixel color	The color of a pixel on the display.
point	To position the tip of the Pointer over an object on the display.
Pointer	The thing that moves on the display when you move the mouse. You use the Pointer to (1) select icons and gadgets (2) choose menu items.
Pointer Editing Window	The window displayed by Preferences in which you change the Pointer.
pop-up requester	A requester that you open by double-clicking the Menu button.
port	A connector for attaching add-ons to the Amiga.
Preferences	A tool that allows you to change various settings of an Amiga, including the time, the Workbench font, the speed that keys on the keyboard repeat when you hold them down, and the interval before keys begin repeating.
press	To push down a mouse button or key on a keyboard.

Printer Requester	A requester provided by Preferences that you use to change printer settings.
project	A place where information created or used by a tool is kept. An example of a project is a note you write with the Notepad .
Project disk	A disk used to store projects .
Project List	The list of projects you can open from an Open Requester .
protect	To prevent the contents of a project , tool , drawer , or disk from being changed.
protect tab	A plastic tab on a microdisk that, when you slide it so that there is a hole through the disk, prevents the information on that disk from being changed.
protected disk	A disk whose contents cannot be modified.
RAM = random-access memory	
random-access memory	Memory whose contents can be changed while the computer is operating.
redraw	To redisplay what appears in a screen .
release	To stop pressing or holding down a mouse button .
rename	To change the name of a tool , project , disk , or drawer .
Repeat Delay Slider	A gadget provided by Preferences for changing how long it takes for a key on the keyboard to repeat when you hold it down.
requester	A rectangular region in a screen which you use to give information to a tool . When a requester appears, you must select a gadget in the requester to close the requester before you can do anything else in the window in which the requester appears.

reset	To set up the Workbench again after it has begun working.
resolution	On a video display , the number of pixels that can be displayed in the horizontal and vertical directions.
reverse video	Displayed using colors opposite those normally used. For example, if letters are normally black on a white background, white letters on a black background are said to be shown in reverse video.
RGB connector	The connector on the back of the main unit that you use to attach an RGB monitor to the Amiga.
RGB monitor	A video monitor , such as the Amiga Monitor , that interprets signals for red, green, and blue to create colors.
row	A set of adjoining pixels that form a horizontal line on the video display .
save	To copy the contents of a project onto a disk .
screen	A full-width area of the video display with the same color palette , resolution , and other attributes.
scroll	To move the contents of a project within a window .
Scroll Arrows	Arrows at both ends of a Scroll Bar . To move slowly forward through a project , put the Pointer over the bottom Scroll Arrow and hold down the Selection button . To move slowly backward through a project, put the Pointer over the top Scroll Arrow and hold down the Selection button .
Scroll Bar	A gadget you use to display different parts of a project .
Scroll Box	The rectangular area within a Scroll Bar that you drag to move rapidly from one part of a project or list to another.

submenu	An additional menu that appears to the side of a menu.
submenu title	An item in a menu that, when you place the Pointer over it, causes a submenu to appear.
select	To pick an icon, gadget , or a part of a project using the Selection button.
Selection button	The left-hand button on the mouse .
selected option	An option that is currently in effect.
selected window	The window that you do work in. Only one window can be selected at a time.
selection shortcut	A quick way to select something by pressing a key on the keyboard while holding down the left Amiga key.
serial port	A connector on the back of the Amiga which you use to attach modems and other add-ons .
set up	To start the Workbench .
shortcut	A quick way, from the keyboard , to (1) choose a menu item (2) select an icon or gadget . See menu shortcut and selection shortcut .
size	To change the dimensions of a window or screen .
Sizing Gadget	A gadget you drag to change the size of a window .
slider	A gadget you use to pick a value within a range, normally by dragging an arrow along a line.
source disk	When copying disks, the disk that is being copied. <i>Compare</i> destination disk.
Space Bar	The long key at the bottom of the keyboard that you press to enter a blank space.
status	A characteristic of a tool , project , drawer , or disk , such as whether it is deletable or not deletable.

string	A set of one or more characters.
String Gadget	A gadget you use to enter or modify strings .
Text Cursor	In projects containing text, a marker that indicates your position in the project.
timesaver	Any technique provided by a tool to save you time. A shortcut is one kind of timesaver; another is double-clicking the Menu button to get a pop-up requester .
Title Bar	A strip at the top of a screen or window that contains the name of the screen or window.
Title Gadget	A gadget in Open Requesters that you use to type in the title of the project you want to open.
tool	A facility for working with information. For example, the Graphicraft tool lets you create and change visual information that takes the form of a painting .
Trashcan	The place where you put projects , tools , and drawers to discard them.
TV modulator	A device used to connect a television set to an Amiga.
TV modulator cable	A cable you use to connect the TV modulator to the TV switch box .
TV switch box	A device that allows you to connect both an Amiga and an antenna to a television and switch between them.
type	The kind of object (tool , project , drawer , or disk) an object is.
type font	A set of letters, numbers, and symbols that are the same type size and of the same typeface .
type size	The size of text.
type style	A variation of a typeface , such as italic or bold.

typeface	A set of letters, numbers, and symbols that share the same design.
unavailable menu item	Any item in a menu that you cannot choose . Unavailable menu items are shown as ghost items .
video cable	The cable you use to connect an RGB monitor to an Amiga.
video equipment	A video monitor or television.
video monitor	A device for displaying visual information from an Amiga.
Wait Pointer	A special shape for the Pointer that indicates that you must wait before continuing.
window	A rectangular area in a screen . Tools use windows to accept and present information.
Workbench	A tool you use to get and manipulate the facilities of the Amiga. You use the Workbench to open , close , move , create, and delete projects , tools and drawers , to copy disks , as well as to perform other operations.
Workbench disk	A disk that contains the Workbench.
Workbench screen	The screen used by the Workbench and other tools.
working disk	A copy of an original disk that came with the Amiga or with a tool.
"Y" adapter	An adapter that lets you combine both of the audio signals from an Amiga into a single audio signal. You use this adapter to connect the Amiga Monitor to an Amiga.

Index

- AC power cord 2-1
- adding to the Amiga 5-1
- add-ons 5-2
 - attaching a printer 7-6
 - cables for 5-2
 - precautions for 5-2
- alerts 4-30
- ALT key
 - using menus with 3-14
- Amiga
 - adding to 5-1
 - assembling 2-1
 - caring for 6-1
 - specifications for 7-18
- Amiga Basic 1-3
- Amiga External 3.5 Disk Drive 5-3
 - connector for 2-2
- Amiga Hardware Manual* 1-3
 - information about connectors in 7-10
- Amiga key, left
 - selecting with 3-10
- Amiga keys
 - moving Pointer with 3-8
- Amiga Memory Expansion Cartridge 5-2
- Amiga ROM Kernel Manual* 1-3
- Amiga Service Centers 6-2
- AmigaDOS A-10
- AmigaDOS Developer's Manual* 1-3
- AmigaDOS errors A-10
- AmigaDOS messages A-10
- AmigaDOS Technical Reference Manual* 1-3
- AmigaDOS User's Manual* 1-3, A-10

- Aspect (printer setting) 7-9
- assembling the Amiga 2-1
- attaching a printer 7-6
- attributes, video 4-3
- audio connections
 - to stereos 2-10
 - to monitors 2-11
- audio connectors 2-10
 - illustration of 2-2
- audio equipment
 - attaching an Amiga to 2-10
- Back Gadget
 - for windows 4-21
 - for screens 4-28
- baud rate, setting 7-5
- cables
 - attaching to connectors 2-2
 - proper cables for add-ons 5-2
 - shielded 5-2, 7-18
 - warning about 2-3
- care of monitors 6-2
- caring for the Amiga 6-1
- cautions *See* warnings
- centering the monitor display 7-4
- Change Printer Screen 7-6
- changing the Pointer A-12
- choosing menu items 4-9
- cleaning the mouse 6-3
- Clean Up 4-32

CLI

- icon for 7-4
- making the CLI available 7-4
- clicking 3-9
- Clock A-2
- Close 4-24
- Close Gadget
 - using 4-24
- closing windows 4-24
- color printers 5-3
- colors
 - changing Workbench colors 7-5
 - for screens 4-4
- Command Line Interface *See* CLI
- commands in menus 4-10
- composite video *See* NTSC
- connecting audio equipment 2-10
- connectors 2-2
- connectors
 - audio 2-2
 - care of 6-2
 - disk drive 2-2
 - keyboard 2-4
 - NTSC 2-2
 - parallel 2-2
 - pin assignments for 7-10
 - RGB 2-2
 - serial 2-2
 - warning about 2-3
- controlling the Workbench 4-4
- copying disks 4-31
 - making backup copies 6-5
- creating a project 3-18
- cursor keys
 - moving Pointer with 3-8
- date and time, changing 7-2
- Demos drawer A-1
- demonstrations of graphics A-1
- diagnostic messages, displaying 4-33
- Discard 4-16, 4-17
- discarding drawers 4-17
- discarding projects 4-15, 4-16
- discarding tools 4-15, 4-16

disk drives

- adding a disk drive 5-3
- using 3-4
- disk drive light 3-5
 - warnings about 3-5, 6-5
- disk gauge 4-25
- disks
 - as drawers 4-17
 - care of 6-5
 - copying 4-31
 - disk gauge 4-25
 - duplicating 3-14, 4-31
 - freeing disk space 4-25
 - how to insert 3-4
 - icons for 4-3
 - initializing 4-30
 - making backup copies 6-5
 - opening 3-13
 - operations involving 4-30
 - preparing new disks 4-30
 - properties of 4-17
 - protect tabs on 3-2
 - reclaiming disk space 4-25
 - removing 3-6
 - renaming 4-32
 - storage in 4-25
 - using 3-2
 - working disks 3-14
- display
 - changing size of text on 7-4
 - centering 7-4
- Display Centering Gadget 7-4
- dot-matrix printers 5-3
- double-click
 - changing the delay for 7-4
- double-clicking
 - to open a tool 3-20
- Drag Bar
 - for windows 4-19
- dragging 4-8
- dragging screens 4-26
- dragging windows 4-20
- drawers 4-16
 - creating 4-17

- Demos A-1
 - discarding 4-17
 - duplicating 4-17
 - Empty 4-17
 - icons for 4-3
 - moving 4-16
 - moving to a new disk 4-32
 - operations involving 4-16
 - renaming 4-17
 - Utilities 3-18
- Duplicate 4-14
 - duplicating drawers with 4-17
- duplicating disks 3-14, 4-31
- duplicating drawers 4-17
- duplicating projects 4-14
- duplicating tools 4-14
- Empty Trash 4-17
- error messages
 - displaying 4-33
- error numbers A-10
- exit gadgets 4-30
- Extended Selection 4-7
- external disk drives 5-3
- Extras disk 3-2
- FCC requirements Copyright Page, 5-2, 7-17
- Federal Communications Commission *See* FCC
- freeing disk space 4-25
- Front Gadget
 - for windows 4-19
 - for screens 4-27
- gadgets
 - Back Gadget 4-19, 4-28
 - Close Gadget 4-24
 - Display Centering Gadget 7-4
 - Drag Bar 4-19
 - exit 4-30
 - for Preferences 3-18
 - Front Gadget 4-19, 4-27
 - ghost 4-19
 - in windows 4-18
 - Scroll Bar 4-19
 - Sizing Gadget 4-19
 - unavailable 4-19
- game controller connectors
 - pin assignments for 7-15
- ghost gadgets 4-19
- ghost menu items 4-11
- glossary G-1
- Graphicraft 3-1
 - manual for 1-3
- graphics, demonstrations of A-1
- highlighting 3-10
- humidity, acceptable range for 7-19
- icons
 - dragging 4-8
 - for Workbench disk 3-6
 - highlighted 3-10
 - on the Workbench 3-6
 - selecting 3-9
 - straightening up 4-32
- Image (printer setting) 7-9
- Info 4-15
- Initialize 4-30
- initializing disks 4-30
- ink-jet printers 5-3
- input/output connectors 7-10
- interfaces *See also* connectors
 - list of 7-19
- interference, radio and television 5-2, 7-17
- Intuition: The Amiga User Interface* 1-3
- key repeat speed, changing 7-3
- keyboard 2-1
 - attaching 2-4
 - care of 6-2
 - changing the tilt of 2-5
 - folding legs on 2-5
 - illustration of 2-2
 - storing under main unit 2-5
- keyboard cable 2-1
 - warning about 6-2
 - illustration 2-2
- Kickstart disk 3-2
 - inserting 3-4

Last Error 4-33
Left Margin (printer setting) 7-7
letter-quality printers 5-3
light pen, connector for 7-15
load limit for main unit 6-2

magnets, precautions about 6-2, 6-5
main unit 2-1

weight it can support 6-2
margins for printing, specifying 7-7
memory 3-16

use in multitasking 4-14

memory meter 4-28

Menu Bar 3-11

Menu button 3-11

menu shortcut 4-12

menu titles 3-12

menus

browsing through 3-13

choosing more than one item 4-12

commands in 4-10

ghost items in 4-11

Menu Bar 3-11

Multiple Choice 4-12

options in 4-10

showing menu titles 4-8

titles of 3-11

using 3-11

using without a mouse 3-14

unavailable items in 4-11

messages, AmigaDOS A-10

messages, error

displaying errors 4-33

meter, memory 4-28

microdisks *See* disks

modulator, TV 2-9

monitors

composite video 2-7

NTSC 2-7

RGB 2-7

using a television as a monitor 2-7

attaching 2-7

care of 6-2

centering the display 7-4

changing size of text displayed on 7-4
differences between 2-7
number of characters displayed 2-7
sound connections for 2-11

mouse 2-1

attaching 2-6

buttons 3-9

care of 6-2

changing mouse speed 7-3

cleaning 6-3

illustration of 2-2

Menu button on 3-11

moving the Pointer without a mouse 3-8

pin assignments for connectors 7-15

room required for 2-6

running out of room for the mouse 3-8

selecting without a mouse 3-10

Selection button on 3-9

using the Amiga without a mouse 4-13

mouse ball 2-6, 6-3

mouse buttons

clicking 3-9

mouse connectors

pin assignments for 7-15

mouseless operation 4-13

selecting 3-10

moving drawers 4-16

to a new disk 4-32

moving projects 4-16

to a new disk 4-32

moving screens to the back 4-28

moving tools 4-16

to a new disk 4-32

moving windows to the front 4-21

Multiple Choice 4-12

multitasking 4-14

Notepad A-4

introduction to 3-18

NTSC connector 2-9

on televisions 2-7

NTSC monitors

connecting 2-9

text size on 7-4

- on/off switch 3-3
- Open 3-18
- opening disks 3-13
- opening projects 4-14
- opening tools 3-18, 4-14
 - by double-clicking the Selection button 3-20
- operating temperature 6-2
- operations involving disks 4-30
- options, menu 4-10

- Paper Size (printer setting) 7-7
- Paper Type (printer setting) 7-7
- parallel connector
 - pin assignments for 7-13
- pin assignments for connectors 7-10
- Pitch (printer setting) 7-8
- pixels
 - number of in screens 4-3
- Plug 'n Print Cartridge 5-3
- plugging in the Amiga 2-13
- point (of the Pointer) A-13
- Pointer
 - changing A-13
 - changing Pointer speed 7-3
 - editing A-13
 - moving 3-7, 4-5
 - moving without a mouse 3-8
 - point of 4-5
 - Wait Pointer 4-6
- pointing 3-7, 4-5
- ports *See* connectors
- power cord
 - illustration of 2-2
- power, requirements for 7-19
- precautions *See* warnings
- precautions for add-ons 5-2
- Preferences 7-2
 - changing the Pointer with A-12
 - getting back settings 7-10
 - introduction to 3-17
 - saving settings 7-10
 - settings for 7-2
 - using settings 7-10
- preparing disks for use 4-30

- Printer Graphics Screen 7-8
- printers
 - adding 7-6
 - attaching 7-6
 - installing 7-6
 - list of supported printers 5-3
 - Preferences settings for 7-6
 - properties of 3-18
 - specifying parallel or serial 7-7
- projects
 - creating 3-18
 - discarding 4-15
 - duplicating 4-14
 - getting information about 4-15
 - moving 4-16
 - moving to a new disk 4-32
 - opening 4-14
 - renaming 4-15
 - reopening a project 3-21
 - saving 3-21
- protect tabs 3-2
- protecting disks 3-2
- pushing windows to the back 4-21
- pushing screens to the back 4-28
- putting together the Amiga 2-1

- Quality (printer setting) 7-7

- radio and television interference 5-2
- RAM
 - adding RAM to the Amiga 5-2
 - meter for available RAM 4-28
- random-access memory *See* RAM
- RGB monitor
 - attaching 2-8
- reclaiming disk space 4-25
- Redraw 4-33
- redrawing the Workbench 4-33
- removing disks 3-6
 - warning about 3-5
- Rename 4-14
 - renaming drawers with 4-17
- renaming disks 4-32
- renaming drawers 4-17

- renaming projects 4-15
- renaming tools 4-15
- requester
 - operations involving 4-29
 - properties of 3-15, 4-29
 - responding to 4-30
- resetting the Workbench 3-16, 4-32
- resolution, screen 4-4
- RGB connector
 - pin assignments for 7-14
- RGB monitors
 - description of 2-7
 - text size on 7-4
- Right Margin (printer setting) 7-7
- screens
 - attributes of 4-4
 - changing colors in the Workbench screen 7-5
 - dragging 4-26
 - moving in front 4-27
 - moving 4-26
 - operations involving 4-25
 - properties of 4-25
 - redrawing 4-33
- Scroll Arrow 4-22
- Scroll Bar
 - for windows 4-22
- Scroll Box 4-22
- scrolling windows 4-22
- selected window 4-18
- selecting
 - Extended Selection 4-7
 - icons 3-9
 - selecting more than one icon 4-7
 - with the mouse 4-6
 - without a mouse 3-10
- Selection button
 - selecting with 3-9
- selection shortcuts 4-12
 - for Workbench screen 4-28
- serial connector
 - illustration of 2-2
- serial port
 - setting the baud rate 7-5
- Service Centers 6-2
- setting the date and time 7-2
- settings, Preferences 3-17
- settings for printers 7-6
- Shade (printer setting) 7-9
- shielded cables for add-ons 5-2
- SHIFT key
 - for Extended Selection 4-7
- shortcuts
 - for Workbench screen 4-28
 - menu 4-12
 - selection 4-12
- size of text, changing 7-4
- Sizing Gadget 4-20
- sizing windows 4-20
- Snapshot 4-33
- Spacing (printer setting) 7-8
- specifications for the Amiga 7-18
- stereo, connecting an Amiga to 2-10
- straightening up icons 4-32
- sunlight, precautions about 6-2
- switch box, TV 2-9
- television
 - attaching to an Amiga 2-9
 - NTSC connectors on 2-7
 - using as a monitor 2-7
- television interference 5-2, 7-17
- temperature, operating 6-2
- text
 - changing size of on the display 7-4
- Textcraft 1-3
- Threshold (printer setting) 7-9
- tilt, keyboard 2-5
- time
 - changing 7-2
 - showing A-2
- Title Bar
 - messages displayed in 4-33
- tools
 - using 3-17
 - Clock A-2
 - demonstration tools 4-33
 - discarding 4-15

- duplicating 4-14
- getting information about 4-15
- Graphicraft 3-1
- icons for 4-3
- moving 4-16
- moving to a new disk 4-32
- Notepad A-4
- opening 4-14
- renaming 4-15
- Trashcan
 - as a drawer 4-17
 - discarding drawers using 4-17
 - emptying 4-17
 - freeing disk space with 4-25
 - getting items out 4-15
 - icon for 4-3
 - properties of 4-17
- turning on the Amiga 3-3
- TV modulator
 - attaching a television with 2-9
- TV Modulator connector
 - pin assignments for 7-15
- TV switch box
 - attaching a television with 2-9
- unavailable menu items 4-11
- using a tool 3-17
- using disks 3-2
- using menus 3-11
- using the Workbench 4-1
- ventilation slots 2-13
- video attributes 4-3
- video cable
 - attaching RGB monitor with 2-8
- video monitor *See* monitor
- Wait Pointer 4-6
- warnings
 - about add-ons 5-2
 - about assembling the Amiga 2-3
 - about cables 6-2
 - about connectors 7-11
 - about copying disks 4-31
 - about discarding tools and projects 4-16
 - about discarding drawers 4-17
 - about initializing disks 4-30
 - about magnets 6-2, 6-5
 - about moisture 6-1
 - about removing disks 3-5
 - about resetting 3-16, 4-32
 - about sunlight 6-2
 - about the case 6-2
 - about the keyboard 6-2
 - about turning on the Amiga 3-3
- warranty information 2-1
- windows
 - closing 4-24
 - components of 4-3
 - dragging 4-20
 - for tools 3-17
 - gadgets in 4-18
 - moving to the front 4-21
 - opening 3-13
 - operations involving 4-18
 - overlapping 4-18
 - properties of 4-18
 - pushing to the back 4-21
 - scrolling 4-22
 - selecting 4-18
 - sizing 4-20
- Workbench
 - changing the colors 7-5
 - control techniques 4-4
 - illustration of 3-6
 - redrawing the screen 4-33
 - resetting 4-32
 - title bar for 3-6
 - using 4-1
- Workbench disk 3-2
 - inserting 3-6
- Workbench operations 4-13
- Workbench screen 4-2
 - shortcuts for 4-28
- Workbench tools A-1
- working disks 3-14
- Y adapter 2-11

Introduction to Amiga Appendix Addendum

Your Workbench upgrade disk contains two additional tools that are not on the original Workbench disk, the Calculator and the Icon Editor. Appendix A of the Introduction to Amiga manual tells you how to use the other tools on the Workbench disk. The information explaining the new Calculator and Icon Editor tools is contained in the following pages, to be added to your manual as Appendix D (The Calculator) and Appendix E (The Icon Editor).

PART NUMBER 318-675-04

The Calculator

The Calculator is a standard four-function calculator you can use to add, subtract, multiply, and divide numbers. You can find the Calculator in the Utilities drawer on the Workbench.

Opening the Calculator

You open the Calculator by selecting its icon, then choosing Open from the Workbench menu. When you do, a window for the Calculator appears.

The Calculator Keys

In the Calculator window, each of the Calculator's "keys" is a gadget. When the Calculator window is selected, there are two ways you can "press a key":

- Select the gadget by pointing within it, then clicking the Selection button.
- For all but the <- and +- "keys," type the character or characters shown in the gadget. For example, you can clear the current entry by pressing the C key on the keyboard, then the E key.

The keys for digits, the decimal point, and addition and subtraction are the same as those on other calculators. To multiply, use the * key. To divide, use the / key.

Selecting the CE key clears the current entry, while selecting the CA key clears the current entry and any previous entries.

Pressing the +- key changes the sign of the current entry. If the current entry is a positive number, it is changed to the negative number that corresponds to it. If the entry is negative, it becomes positive.

When you're entering a number, pressing the <- key deletes the last digit you entered.

To get a result, use the = key. From the keyboard, you can get a result by pressing either the = key or the ENTER key.

Closing the Calculator

To close the Calculator, select the Close gadget in the upper left corner of the Calculator window.

The Icon Editor

With the Icon Editor, you can change the appearance of icons that appear on the Workbench. You can find the Icon Editor in the System drawer on the Workbench.

To use the Icon Editor, you must be familiar with the Amiga Disk Operating System (AmigaDOS) and the conventions it uses for file names. To learn about AmigaDOS, see the *AmigaDOS User's Manual*.

If you are a software developer, you can also use the Icon Editor to create icons for new tools, projects, and drawers. To learn how to create new icons, see the *Amiga ROM Kernel Manual*.

Opening the Icon Editor

You open the Icon Editor by selecting its icon, then choosing Open from the Workbench menu. When you do, a window for the Icon Editor appears. Next, a requester appears in the window that describes the different kinds of icons. Select the OK gadget to continue. (For information about the different icon types, see the *Amiga ROM Kernel Manual*.)

Loading an Icon

To select an icon you want to change, first select a frame—one of the nine boxes to the right of the Icon Editor window—by pointing within the frame, then clicking the Selection button. When you first open the Icon Editor, each of the nine frames contains the Icon Editor icon. The icon you select will replace what appears in the currently selected frame. Next, choose Load Data from the Disk menu. In the requester that appears, select the gadget immediately below the words “Enter Icon Name (.info Will Be Added)”, then enter the AmigaDOS description for the file or directory whose icon you want to change. This description can be either:

- the complete AmigaDOS file or directory description.
- an abbreviated description that specifies the relationship of the file or directory to the directory in which the Icon Editor resides.

For example, you can load the icon for the Trashcan by entering either:

```
df0:Trashcan
```

```
/Trashcan
```

NOTE: the icon *types* shown in the requester that appears when you open the Icon Editor are **not** the names of icons. To find the name of a file whose icon you want to change, use the AmigaDOS DIR command.

After you select the gadget, characters you type appear to the left of the Text Cursor (the marker that appears in the gadget when you select it). To move the cursor, use the left and right cursor keys.

There may already be text in the gadget when you select it. You can delete characters at and to the right of the Text Cursor by pressing the DEL key. Press the BACKSPACE key to delete characters to the left of the Text Cursor.

There are shortcuts you can use to change what appears in the gadget and to move the Text Cursor:

- Press the right Amiga key and Q key at the same time to get back what was in the gadget before you selected it.
- Press the right Amiga key and the X key at the same time to erase what appears in the gadget.
- Press the SHIFT key and the left cursor key at the same time to move the Text Cursor to the leftmost character in the gadget.
- Press the SHIFT key and the right cursor key at the same time to move the Text Cursor to the right of the rightmost character in the gadget.

When you've finished entering the file description, select Load Icon Image. If you decide not to select an icon, select Cancel The Load.

Selecting Additional Icons

With the Icon Editor, you can work with up to nine icons at the same time. To select an additional icon, first select the frame in which you want the icon to appear, then choose Load Data from the Disk menu.

Changing an Icon

A magnified view of the currently selected frame is shown at the left of the Icon Editor window. To change the appearance of an icon, you select the frame in which it appears, then change what appears in the magnified view. The techniques for changing the view are described below.

Changing an Icon's Colors

To change the color of an individual pixel in an icon, choose a color from the Color menu, point to the pixel you want to change in the magnified view, then click the Selection button. By holding down the Selection button while you slowly move the mouse, you can add color to larger areas.

Filling Areas

With the Icon Editor's Flood Fill feature, you can fill a contiguous area that is all the same color with another color. To fill an area, choose the color you want to fill with from the Color menu, then choose Flood Fill from the Misc menu. Next, point to the area in the magnified view that you want to fill, then click the Selection button.

Adding Text to an Icon

There are six steps to adding text to an icon:

1. Choose Write Into Frame from the Text menu.
2. In the requester that appears, select the gadget immediately below the words "Icon Text," then enter up to eight characters that you want to add to the icon. The characters you type appear to the left of the Text

Cursor (the marker that appears in the gadget when you select it). To move the cursor, use the left and right cursor keys.

There may already be text in the gadget when you select it. You can delete characters at and to the right of the cursor by pressing the DEL key. Press the BACKSPACE key to delete characters to the left of the cursor.

There are shortcuts you can use to change what appears in the gadget and to move the Text Cursor:

- Press the right Amiga key and Q key at the same time to get back what was in the gadget before you selected it.
 - Press the right Amiga key and the X key at the same time to erase what appears in the gadget.
 - Press the SHIFT key and the left cursor key at the same time to move the Text Cursor to the leftmost character in the gadget.
 - Press the SHIFT key and the right cursor key at the same time to move the Text Cursor to the right of the rightmost character in the gadget.
3. Select the foreground and background colors for the text. (How the foreground and background colors are used to display text depends on the display mode you select. Display modes for text are described below.) To change the foreground color, point within the color immediately to the right of the word Foreground, then click the Selection button one or more times until the color you want appears. Select the background color in the same way.
 4. There is only font you can choose for your text. This font, called Topaz, is the same one used by the Workbench for menus and icons. You can, however, select one of two font sizes: TOPAZ_SIXTY is the larger, TOPAZ_EIGHTY the smaller. If the size you want does not

appear in the gadget labeled Font, select the gadget to change to the other size.

5. Select one of the four display modes for text. These modes are:
 - JAM1, where text is shown in the currently selected foreground color without a background
 - JAM2, where text is shown in the foreground color against the currently selected background color
 - COMPLEMENT, where each pixel that makes up the text is the color "opposite" the color of the pixel that it replaces. (To see how this works, add text to an icon containing all four colors, then move the text as described below.)
 - INVERSVID, where the text is surrounded by the currently selected foreground color. When you add INVERSVID text to an icon, the background for the text replaces existing pixels in the icon while the text itself does not.

To change the display mode, point within the gadget labeled Mode, then click the Selection button one or more times until the name of the mode you want appears. In the gadget, note that the name of the mode is displayed in the gadget using the currently selected foreground color, background color, and display mode.

6. Select the Position gadget to add the text to the icon. In the requester that appears, select the arrows to move the text up, down, left, or right. Select the Single gadget if you want the text to move only a single pixel each time you select an arrow. Select Repeat if you want the text to continue to move if you hold down the Selection button after you select an arrow. When the text is where you want it, select OK. If you change your mind, select Cancel to return to the previous requester without adding text to the icon.

When you're through adding text, select OK. If you want to start over with the icon as it was before you chose Write Into Frame, select Reset. If you decide you don't want to add text to the icon, select Cancel.

Using the Undo Feature

If you think that a change you're about to make to an icon may not turn out the way you want, choose Snapshot Frame from the Copy menu before you make the change. When you do, the Icon Editor saves a copy of the currently selected frame. If things go wrong, you can get back what you had before by choosing Undo Frame from the Copy menu.

NOTE: When you choose Undo Frame, the frame that was saved when last chose Snapshot Frame replaces the **currently selected frame**. After you choose Undo Frame, **the previous contents of the currently selected frame are no longer available**.

Working with Frames

The Icon Editor's nine frames let you work with more than one icon at the same time. You can also use the frames to keep and compare more than one version of the same icon. Listed below are techniques you can use when working with frames.

Copying a Frame

To copy a frame, first select the frame into which you want to make the copy. Next, choose the frame you want to copy from the submenu that appears when you point to From Frame in the Copy menu.

Moving the Image within a Frame

To move the image within a frame, choose In-Frame from the Move menu. In the requester that appears, select the arrows to move the image up, down, left, or right. Select the Single gadget if you want the image to move only a single pixel each time you select an arrow. Select Repeat if you want the image to continue to move if you hold down the Selection button when you select an arrow.

The square gadget surrounded by the arrows is the Restore gadget. Select this gadget to put the image back where it was before you chose In-Frame.

When the image is where you want it, select OK. If you change your mind, select Cancel to get back what you had before you chose In-Frame.

Exchanging Frames

To switch the positions of two frames, first select one of the two frames. Next, choose the other frame from that submenu that appears when you point to Exchange With Frame in the Move menu.

Merging Frames

To combine the contents of two frames, first select one of the frames. (The combination you create will replace what appears in this frame.) Next, choose the other frame from the submenu that appears when you point to Merge With Frame in the Copy menu.

When pixels in the two frames overlap, the color that appears is determined as follows:

- If color 0 (the color at the top of the Color menu) overlaps with any other color, the other color is displayed.

- If color 3 (the color at the bottom of the Color menu) overlaps with any other color, color 3 is displayed.
- If colors 1 and 2 (the two colors in the middle of the Color menu) overlap, color 3 (the color at the bottom of the Color menu) is displayed.

Highlighting an Icon

When you select an icon on the Workbench, it is highlighted to indicate that it's selected. There are two ways an icon can be highlighted:

- It can be shown in inverse video. In inverse video, any part of an icon normally shown using color 0 (the color at the top of the Color menu) becomes color 3 (the color at the bottom of the Color menu); color 1 (the color just below color 0 in the menu) becomes color 2; color 2 becomes color 1; color 3 becomes Color 0.
- It can be "backfilled." A backfilled icon is the same as an icon shown in inverse video, with one exception: contiguous areas of an icon normally shown in color 0 that adjoin any of the borders of the icon remain color 0 when the icon is highlighted.

To highlight an icon in inverse video, choose Inverse from the HiLite menu before you save the icon. To backfill a highlighted icon, choose Backfill from the HiLite menu before you save the icon.

Specifying the Border Width

Below each icon on the Workbench is the name of the file that the icon represents. The Icon Editor lets you choose either to put one blank line between the icon's image and its name or not to put space between the icon the image. From the Misc menu, choose either 0 (for no space) or 1 (for

one line between the image and the name) from the submenu that appears when you point to the Set Bottom Border item.

Saving an Icon

When you save an icon, you replace an icon on the Workbench with the icon in the currently selected frame. However, the type of the icon you replace and the type of the icon that was most recently loaded into the currently selected frame must be the same. There are five types of icons:

TYPE	REPRESENTS	EXAMPLE
<i>Disk</i>	Disk drawers	Workbench disk icon
<i>Drawer</i>	Drawers other than disk drawers	System icon
<i>Tool</i>	Tools	Icon Editor icon
<i>Project</i>	Projects	Icon for a Notepad note
<i>Garbage</i>	A drawer that cannot be moved to another drawer	Trashcan icon

When you open the Icon Editor, the Icon Editor icon is loaded into all nine frames. Because the Icon Editor icon represents a tool, you must load another icon if you want to replace an icon of another type.

There are five steps to saving an icon:

1. Select the frame containing the icon.

2. Choose Save Data from the Disk menu. In the requester that appears, select the gadget immediately below the words "Enter Icon Name (.info Will Be Added)", then enter the AmigaDOS description for the file or directory whose icon you want to replace. This description can be either:

- the complete AmigaDOS file or directory description
- an abbreviated description that specifies the relationship of the file or directory to the directory in which the Icon Editor resides. For example, you can replace the icon for the Trashcan by entering either:

```
df0:Trashcan
```

```
/Trashcan
```

After you select the gadget, characters you type appear to the left of the Text Cursor (the marker that appears in the gadget when you select it). To move the cursor, use the left and right cursor keys.

There may already be text in the gadget when you select it. You can delete characters at and to the right of the Text Cursor by pressing the DEL key. Press the BACKSPACE key to delete characters to the left of the Text Cursor.

There are shortcuts you can use to change what appears in the gadget and to move the Text Cursor:

- Press the right Amiga key and Q key at the same time to get back what was in the gadget before you selected it.
- Press the right Amiga key and the X key at the same time to erase what appears in the gadget.
- Press the SHIFT key and the left cursor key at the same time to move the Text Cursor to the leftmost character in the gadget.

- Press the SHIFT key and the right cursor key at the same time to move the Text Cursor to the right of the rightmost character in the gadget.
3. When you've finished entering the file description, there are two ways to save the icon:
- If you want the icon to include the entire image that's in the frame, select Save Full Image.
 - If you want the icon to include only a part of the image that's in the frame, select Frame and Save. Next, "frame" within a rectangle the part of the magnified view you want to include in the icon: point to a place in the magnified view where you want the top left corner of the rectangle, then click the Selection button. Move the mouse to change the size of the rectangle. When you've framed within the rectangle the part the image you want, click the Selection button a second time to save the icon. (If you change your mind about saving the icon after you've selected the top left corner, move the Pointer outside the magnified view and click a mouse button.)

If you decide not to save an icon after you've chosen Save Data, select Cancel the Save in the requester.

When you look on the Workbench for an icon you've saved, remember that it doesn't replace the previous icon until the next time you open the drawer that contains it.

Stopping the Icon Editor

When you are finished using the Icon Editor, close it by selecting the Close Gadget in the upper left corner of the Icon Editor window.



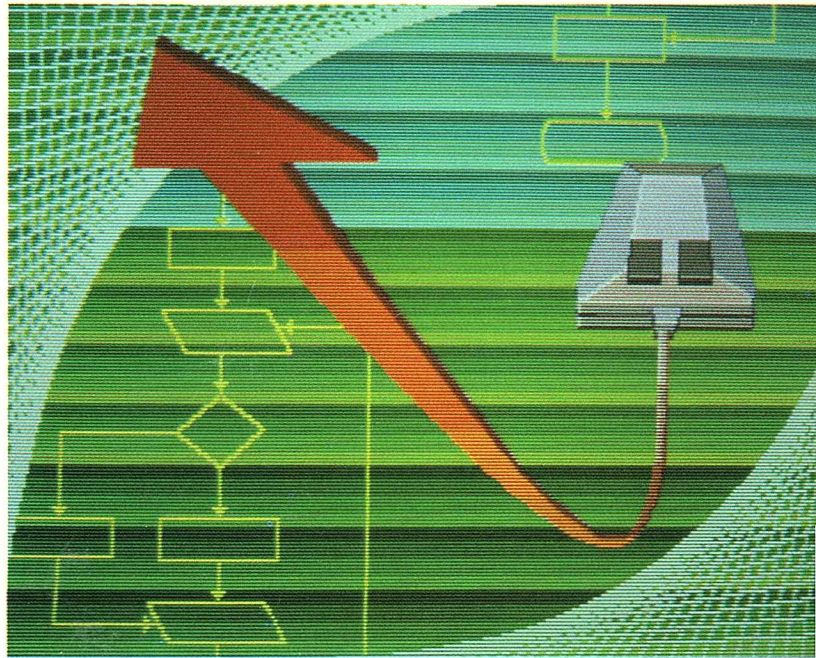
Commodore Business Machines, Inc.
1200 Wilson Drive, West Chester, PA 19380

Commodore Business Machines, Limited
3370 Pharmacy Avenue, Agincourt, Ontario, M1W 2K4

Copyright 1985 Commodore-Amiga, Inc.

AMIGA™

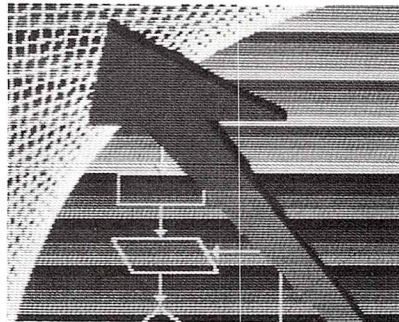
Amiga Basic



Microsoft® BASIC for the Amiga

AMIGA

Amiga Basic



Amiga Basic was developed by Microsoft Corporation.

Microsoft® BASIC for the Amiga

COPYRIGHT

This manual Copyright © Commodore-Amiga, Inc. and Microsoft Corporation, 1985, All Rights Reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from Commodore-Amiga, Inc.

This software Copyright © Microsoft Corporation, 1985, All Rights Reserved. The distribution and sale of this product are intended for the use of the original purchaser only. Lawful users of this program are hereby licensed only to read the program, from its medium into memory of a computer, solely for the purpose of executing the program. Duplicating, copying, selling, or otherwise distributing this product is a violation of the law.

DISCLAIMER

THE PROGRAM IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE PROGRAM IS ASSUMED BY YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU (AND NOT THE DEVELOPER OR COMMODORE-AMIGA, INC. OR ITS DEALERS) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. FURTHER, COMMODORE-AMIGA DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF, OR THE RESULTS OF THE USE OF, THE PROGRAM IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE; AND YOU RELY ON THE PROGRAM AND THE RESULTS SOLELY AT YOUR OWN RISK. IN NO EVENT WILL COMMODORE-AMIGA, INC. BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE PROGRAM EVEN IF IT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME LAWS 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.

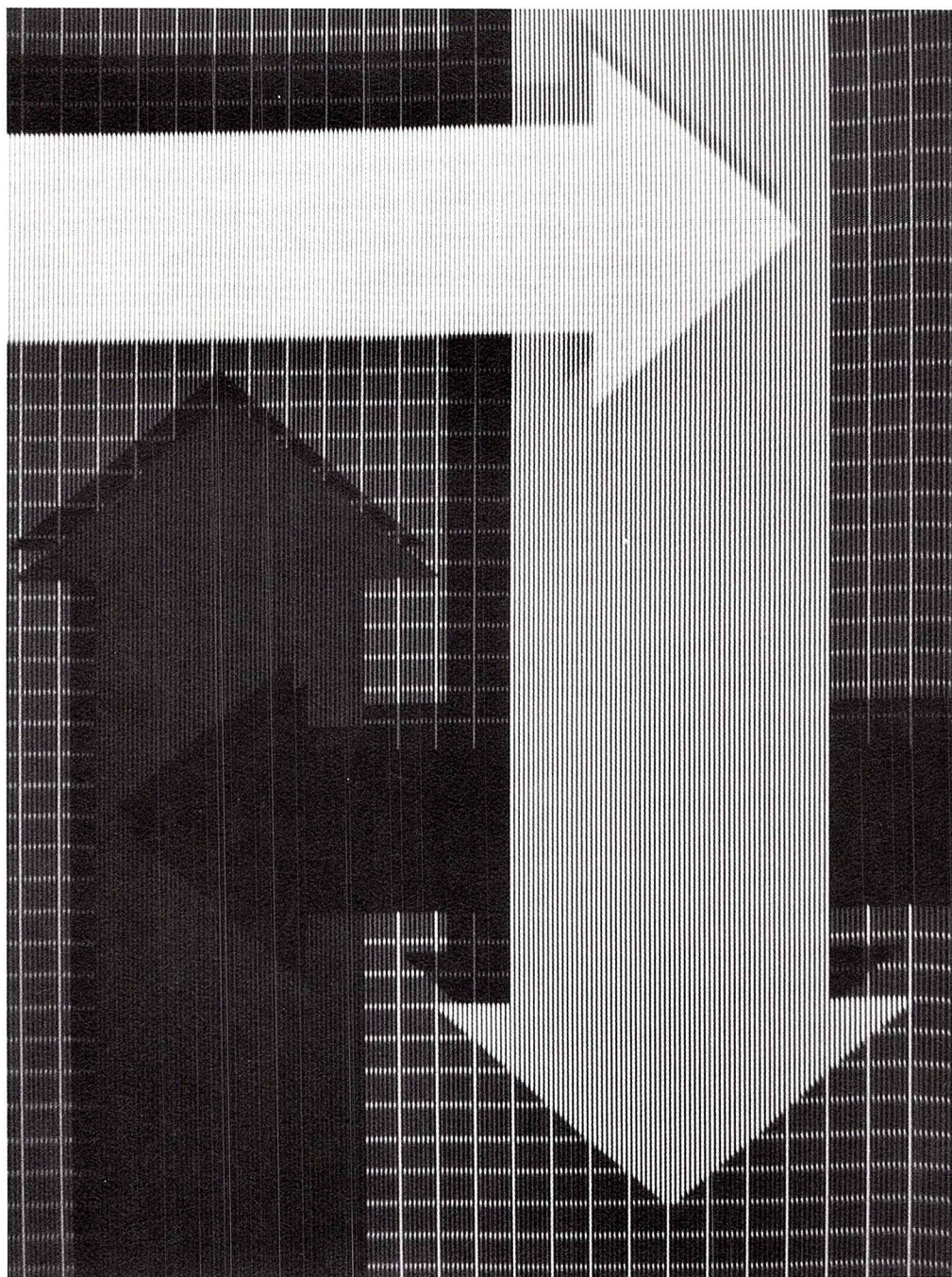
Microsoft is a registered trademark of Microsoft Corporation.
Amiga is a trademark of Commodore-Amiga, Inc.

PRINTED in U.S.A.

CBM Product Number 327273-01 Rev A

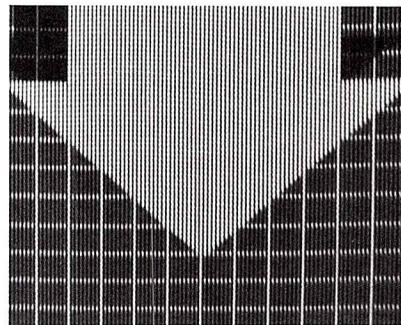
Contents

Chapter 1: Introducing Amiga Basic	1-1
Chapter 2: Getting Started	2-1
Chapter 3: Using Amiga Basic	3-1
Chapter 4: Editing and Debugging Your Programs	4-1
Chapter 5: Working with Files and Devices	5-1
Chapter 6: Advanced Topics	6-1
Chapter 7: Creating Animated Images with the Object Editor	7-1
Chapter 8: BASIC Reference	8-1
Appendices	A-1
Index	I-1



Chapter 1

Introducing Amiga Basic



Who uses BASIC? People use the BASIC programming language for many different reasons. Some of these people are professional programmers. Others are not programmers at all, but wish to run BASIC programs they have purchased. Probably the largest segment of BASIC users is made up of people who write BASIC programs for their own use. They may simply enjoy the mental exercise of programming, or they may have special applications for which they cannot buy ready-made programs. Many BASIC users are students who are studying computer science or using a computer to help with their school work.

All of these people have one thing in common. They use BASIC because it is the universal language for small computers. It is easy to learn, readily available, and highly standardized. It is also a versatile language that has been used in the writing of business, engineering, and scientific applications, as well as in the writing of educational software and computer games.

Amiga Basic

Whatever your reason for using BASIC, you will find that Amiga Basic gives you all the well-known advantages of BASIC, plus the ease of use and fun you expect from Amiga tools. Amiga Basic puts the full BASIC language on your Amiga computer, including BASIC statements used to write graphics, animation, and sound programs. Also, it has all the familiar features of the Amiga screen. Amiga Basic has a Menu Bar, a Pointer, and windows and screens, just like other Amiga tools have.

If you are just starting to learn BASIC, either in a class or on your own, Amiga Basic will fit right in with your course of study. Amiga Basic is based on Microsoft BASIC, the most popular programming language in the world, which works on every major microcomputer.

If you are an old hand at BASIC programming, you'll want to try some of the special features of this version of BASIC, such as SOUND and WAVE for making music and sounds, and GET and PUT for saving and retrieving graphics by the screenful.

About This Manual

This book describes the Amiga Basic Interpreter. It assumes you have read *Introduction to Amiga*, and are familiar with menus, editing text, and using the mouse.

The front part of this book (Chapters 1–7) describes how to use Amiga Basic with the Amiga. It includes a few words on getting started and general instructions on using the interpreter, editing and debugging your programs, working with files and devices, some advanced topics, and a guide to using the Object Editor, a program written in BASIC, that lets you create images to use in animations with your application programs. The back part of this book (Chapter 8) is a reference for the BASIC language. Use the BASIC reference section to read about general characteristics of the language, and to look up the syntax and usage of BASIC statements and functions in the Statement and Function Directory.

Special Features of Amiga Basic

The Amiga Basic interpreter is written in assembly language and thus is small (80K). The core of Amiga Basic has been field tested for three years. Amiga Basic is a “standard” BASIC in that it will run most programs that were written in Microsoft BASIC on most other machines.

Ease of Program Development

Like all languages, Amiga Basic is always growing, changing, and improving. Amiga continues to keep its BASIC interpreter up to date with new features. Here are some of the latest features you’ll find in this version of BASIC. All of the features are described fully in the reference section of the manual.

Support for Amiga Application Programs

Amiga Basic provides the tools you need to write programs that work like and look like they were written for the Amiga. These tools are especially important if you are a software developer who plans to sell application programs for the Amiga.

It is also true that significant Macintosh MS-BASIC and IBM-PC BASIC applications can be ported over to the Amiga extremely easily.

Mouse Support

With the MOUSE function, your BASIC program can accept and respond to mouse input. The MOUSE function returns the coordinates of the mouse pointer under various conditions (left button up, left button down, single-click, double-click, and drag).

MENU Statement

Your programs can display Amiga-style menus created by BASIC's MENU statement. This statement opens and closes menus and highlights menu items. If you want, you can replace BASIC's menus with your own menus, to give your program a completely "custom" look.

Powerful Language Features

Amiga Basic provides a number of powerful language features that lend flexibility to your programs. These features include the following:

Block Statements

IF-THEN ELSE statements let your program make decisions during program execution. You can now include multiple statements on one or more lines after THEN.

Subprograms

Amiga Basic allows subprograms that have their own local variables. Using subprograms, you can build a library of BASIC routines that can be used with different programs. You can do this without concern about duplicating variable names in the main program.

SHARED Statement

The SHARED Statement allows variables to be shared between the main program and its subprograms.

Integer Support

Amiga Basic includes both 16 and 32 bit integer support.

Floating Point Support

The Amiga version includes both 32 and 64 bit floating point support.

No Line Numbers Required

Program lines do not require line numbers. Assigning labels to functional blocks lets you quickly see the control points in your program.

Alphanumeric Labels

Alphanumeric line labels beginning with an alphabetical character allow the use of mnemonic labels to make your programs easier to read and maintain.

Sequential and Random Access File Support

Both sequential and random access files can be created. Sequential files are easy to create, while random access files are flexible and quick in locating data.

Device Independent I/O Support of RS232 and Parallel Ports

Using Amiga Basic's traditional disk file-handling statements, a program can direct both input and output from the screen, keyboard, line printer, and RS232 and parallel ports. You can open the line printer or screen for output as easily as you open a disk file.

Features That Show Off The Amiga

A number of features of Amiga Basic enhance Amiga's color, graphics, animation, and sound capabilities:

- Four-voice synchronized musical reproduction through the SOUND and WAVE statements
- Creation of audible speech through the SAY and TRANSLATE\$ statements
- The ability to save and redisplay screen images through the GET and PUT statements
- Full complement of graphic statements, such as LINE, CIRCLE, PAINT, AREA, and AREAFILL
- Extensive animation support through the OBJECT statements, the Object Editor, and the COLLISION function.
- The ability to call subroutines written in machine language through the LIBRARY and DECLARE statements
- Multiple screens and windows through the SCREEN and WINDOW statements.
- Pull-down Menus from BASIC and the application programs

All of these functions are described in detail under the related commands in Chapter 8; the Object Editor is described in Chapter 7. Some of the functions are summarized below.

SOUND and WAVE

Amiga Basic programs can produce high quality sound for games, music applications, or user alerts. The SOUND statement emits a tone of specified frequency, duration, and volume. As an option, the tone can also have one of four user-defined "voices." The WAVE statement lets you assign your own complex waveforms to each of the voices. SOUND and WAVE can provide your programs with a rich variety of musical sounds, from the complexity of a string quartet to the simplicity of a whistled tune.

LINE and CIRCLE

LINE and CIRCLE are versatile commands for drawing precise graphics. The LINE statement draws a line between two points. The points can be expressed as relative or absolute locations. By adding the B option to the LINE statement, you can draw a box. Another option, BF, fills in the box with any color.

The CIRCLE statement draws a circle, arc, or ellipse according to a given center and radius. A color option can be used to draw the circle in any color. Another option, aspect, determines how the radius is measured, so you can adjust it to create a variety of ellipses.

GET, PUT, and SCROLL

The GET statement saves groups of points from the screen in an array, so you can store a "picture" of a graphic image in memory. The PUT statement calls the array back and puts it on the screen. The SCROLL statement lets you define an area of the screen and how much and which way you would like it to move.

The Object Editor

Amiga Basic offers the Object Editor, a program written in BASIC, that helps you create images of objects to use for animations with your Amiga Basic applications programs. See Chapter 7 for details on the Object Editor.

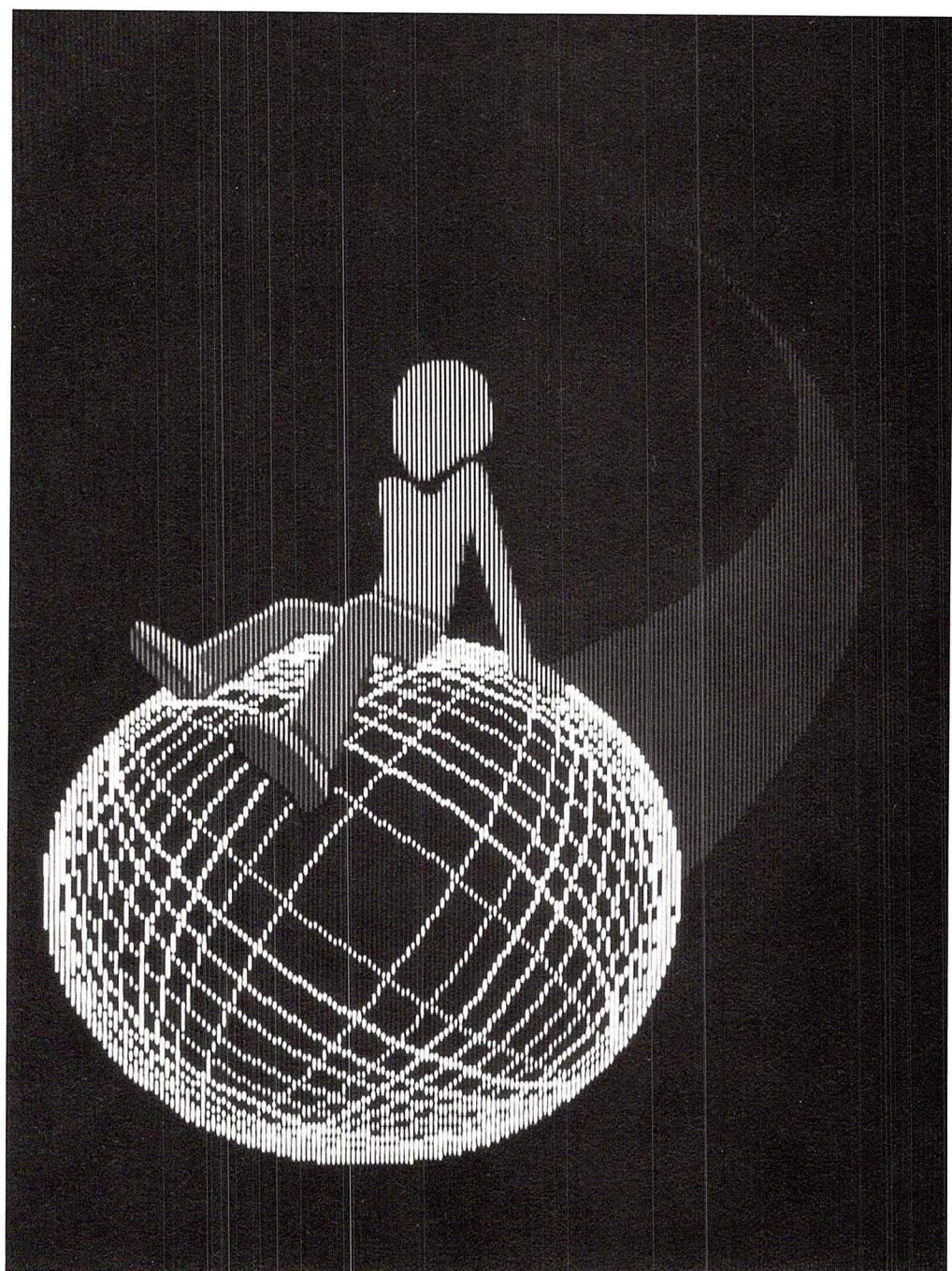
Learning More About BASIC and the Amiga

This manual provides complete instructions for using the Amiga Basic Interpreter. However, little training material for BASIC programming is included. If you are new to BASIC or need help in learning to program, we suggest you read one of the following:

Dwyer, Thomas A., and Critchfield, Margot. *BASIC and the Personal Computer*. Reading, Mass.: Addison-Wesley Publishing Co., 1978.

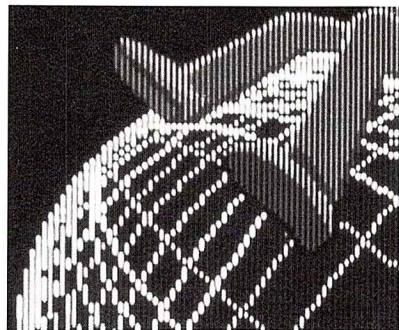
Knecht, Ken. *Microsoft BASIC*. Beaverton, Ore.: Dilithium Press, 1982.

Boisgontier, Jacques, and Ropiequet, Suzanne. *Microsoft BASIC and Its Files*. Beaverton, Ore.: Dilithium Press, 1983.



Chapter 2

Getting Started



To use Amiga Basic, you need:

- An Amiga computer, properly set up and connected.
- The Amiga Extras disk.

You should also make two backup copies of your Amiga Basic disk on your own blank disks. To start Amiga Basic:

- Turn on the Amiga power switch.
- Put the Amiga Extras disk into the Amiga disk drive.
- From the Workbench, open the Amiga Basic icon.

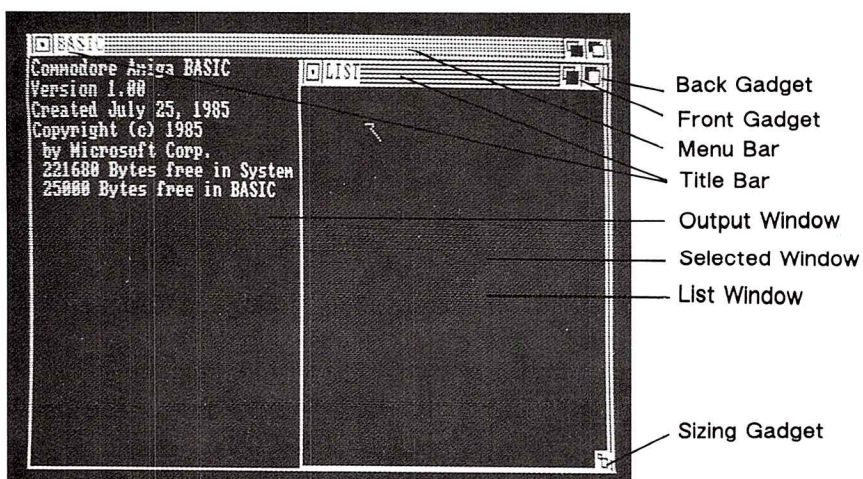
In a few seconds, you'll see the Amiga Basic screen.

Note: This tutorial assumes the Amiga Basic screen is using the original Workbench colors (blue for background, white for foreground, orange, and black).

At this point, the cursor (an orange vertical bar) appears in the List window, and you can either type in a new program or retrieve an existing program and modify it, as you'll see in the next section. Notice that the Title Bar in the List window is displayed distinctively to indicate that it is selected, while the Title Bar in the BASIC window is ghosted or displayed less distinctively to indicate that it is not selected.

The Output window in Amiga Basic not only lets you see the results of a program, it also allows you to type in commands directly. Any time you would prefer to type in commands directly in the Output window, click in the Title Bar of the Output window (entitled BASIC). This process is called selecting the Output window. Notice that BASIC responds with the Ok prompt.

To display the menu titles in the Menu bar, click in the Output window then press and hold down the mouse Menu button.



Practice Session with Amiga Basic

Time Required: Fifteen Minutes

Now you are ready to begin using BASIC.

To display the contents of the Amiga Basic disk in the Output window,

- select the Output window.

When the Ok prompt appears in the window,

- you type

files

and

- press the RETURN key

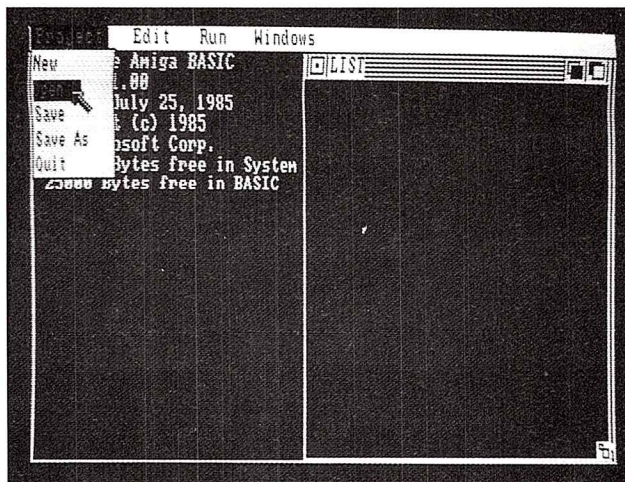
You now see the filenames being listed in the Output window. When the window fills, the names scroll upwards to make room for more names at the bottom of the window. To halt scrolling, press the Amiga command key on the righthand side of the keyboard and the *S* key; to resume scrolling, press any key.

Observe that filenames with the .bas suffix identify the programs you can actually run (called executable programs).

Loading Picture

Start by loading the program called Picture.bas. Picture is a demonstration program, written in BASIC, that comes on your Amiga Basic disk.

- Press the mouse Menu button and point at the Project menu title in the Menu Bar. The menu items that appear are New, Open, Close, Save, Save As, and Quit.
- Choose the Open item.



A requester appears on the Output window.

- Click the mouse Selection button in the Title Gadget labeled "Name of program to load":
- Type

```
picture.bas
```

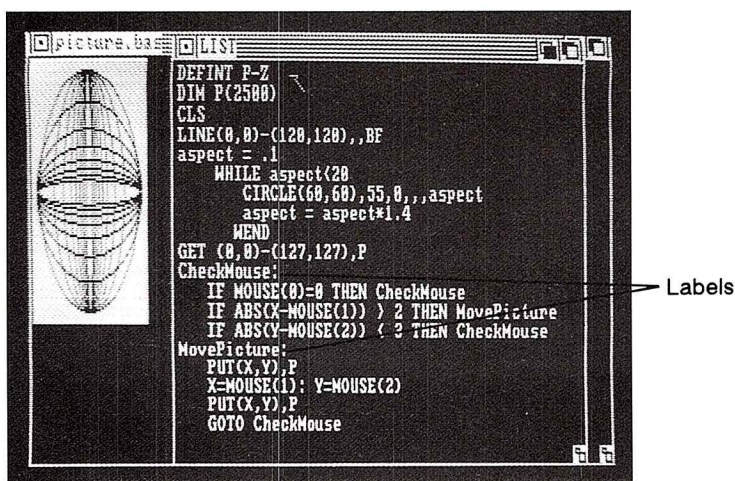
and

- click the OK Gadget or press the RETURN key.

The Program Listing for Picture

A listing of the Picture program appears in the List window. The name of the Output window changes from BASIC to picture.bas.

You may have expected to see a line number at the beginning of each line. In Amiga Basic, line numbers are optional. To refer to a particular line, give that line a label or a line number. For example, the Picture program has no line numbers. However, it has two labels: CheckMouse and MovePicture.



Labels identify entry points

- from GOTO statements executed in other parts of the program, and
- into subsections called from other parts.
- Select the Output window, then type

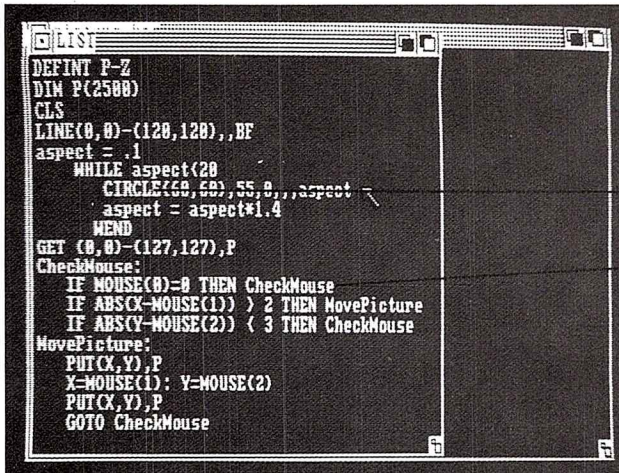
```
list CheckMouse
```

and

- press the RETURN key.

Notice that the List window scrolls to the CheckMouse: label. However, if you wish to edit in the List window, you must first select it.

Uppercase Reserved Words: On the Amiga screen, BASIC program listings are very easy to read because BASIC's reserved words are automatically converted to uppercase as you move from line to line.



```
LIST
DEFINT P-Z
DIM P(2500)
CLS
LINE(0,0)-(120,120),BF
aspect = .1
  WHILE aspect<20
    CIRCLE(50,50),55,0,,,aspect
    aspect = aspect*1.4
  WEND
GET (0,0)-(127,127),P
CheckMouse:
  IF MOUSE(0)=0 THEN CheckMouse
  IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
  IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture:
  PUT(X,Y),P
  X=MOUSE(1): Y=MOUSE(2)
  PUT(X,Y),P
  GOTO CheckMouse
```

Amiga Basic reserved words
are in uppercase

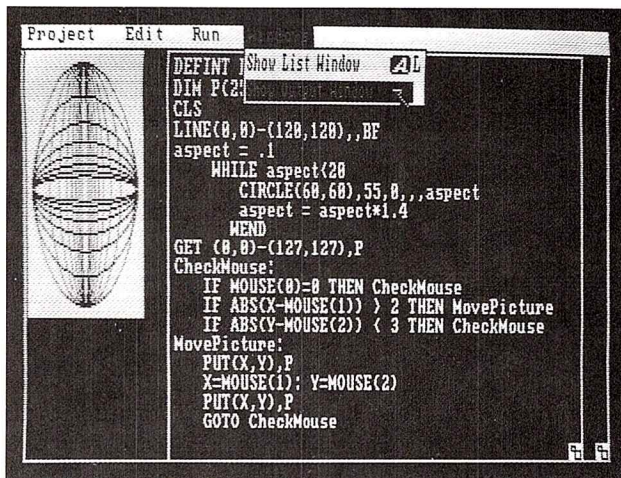
Other words appear as
entered by user

Note that when you type a program line, the reserved word doesn't appear in uppercase until you move from line to line.

What Picture Does

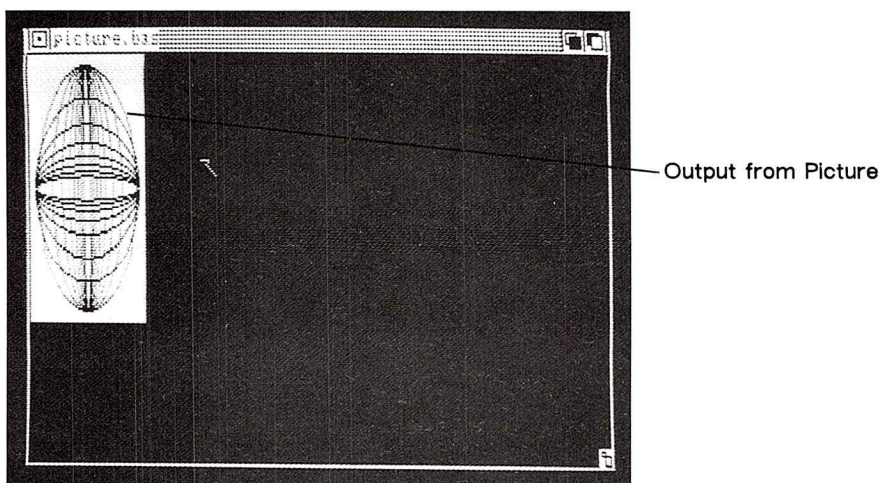
Now, start the program as follows:

- To open the Output window over the List window, choose Show Output Window from the Windows menu.



- Choose Start from the Run menu.

When program runs, a picture appears in the Output window. You can move this picture around by clicking the mouse Selection button anywhere in the Output window. Try it.



Stopping the Program

Picture keeps running until you tell it to stop.

- Choose Stop from the Run menu.
- Choose Show List Window from the Windows menu. The List window comes forward again. To edit the program again in the List window, you must select the List window.

Moving Through the List Window

To scroll through the List window line by line, click in it and use the up and down arrow keys located at the lower right corner of the Amiga screen to move up and down.

To move right or left one character at a time within a program line, use the right or left arrow keys.

To move through the program one window at a time, press the SHIFT key and the up or down arrow at the same time.

Note: Throughout this manual, whenever you see two keys joined together with a hyphen, such as SHIFT-Up Arrow, this means that you press and hold down the first key at the same time that you press the second key. So SHIFT-Up Arrow means to press and hold down the SHIFT key while you press the Up Arrow key.

So, to move forward through the program, window by window, press SHIFT-Down Arrow. To move backward through the program, window by window, press SHIFT-Up Arrow.

To move to the first line in the program, press ALT-Up Arrow. To move to the last line in the program, press ALT-Down Arrow.

To move to the right margin of a program line, press ALT-Right Arrow. To move to the left margin of a program line, press ALT-Left Arrow.

To move 75% through a program line towards the right margin, press SHIFT-Right Arrow. This is convenient for moving through extremely long program lines. To move 75% through a program line towards the left margin, press SHIFT-Left Arrow.

If you want to know more about Picture, see Appendix G, "A Sample Program," for a line-by-line explanation.

Editing a BASIC Program

Editing an Amiga Basic program is similar to editing text with a word processor. You enter and edit all text in the List window using the Cut, Copy, and Paste commands from the Edit menu.

To enter new text, you select the insertion point (the thin orange cursor) by moving the Pointer to the location where you want text, clicking, and typing in the desired characters.

To delete characters to the left of the insertion point, press the BACKSPACE key. To select a block of text, you set the insertion point and drag the mouse.

To select a word, you position the pointer over the word and double-click the mouse Selection button.

To make an extended selection, you can click at the beginning of the selection, move the mouse to the end of the selection, and shift-click (that is, press and hold down the SHIFT key on the Amiga while you click the mouse Selection button. You can Cut or Copy the selected blocks of text just as you would with a word processor.

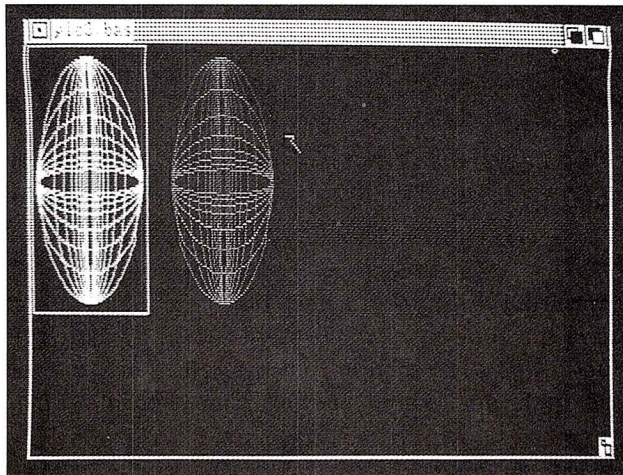
To increase the width of the List window in order to view the entire program listing,

- Press and hold down the mouse Selection button in the Title Bar and drag the entire List window to the left.
- Release the Selection button and move it to the Sizing Gadget on the lower right side. Press and hold down the Selection button over the Sizing Gadget and drag it to make the List window wide enough to read the program lines.
- Release the Selection button when you are satisfied with the List Window width.

Practice Editing with Picture

This is a good opportunity to practice editing a BASIC program on the Amiga and to learn about some of the graphics statements in Amiga Basic. Don't worry about losing or altering Picture. There is another program just like it called picture2.bas on this disk.

If you'd like to experiment, go ahead and make your own changes to Picture. Try the following sequence to change the program to produce output that looks like this:



Adding a Line to the Program

Start by adding the line that draws the second sphere:

- Look in the List window for Picture listing until you find this line:

```
CIRCLE(60,60),55,0,,,ASPECT
```

```

LIST
DEFINT P-Z
DIM P(6000)
CLS
LINE(0,0)-(120,120),,BF
ASPECT = .1
  WHILE ASPECT<20
    CIRCLE(60,60),55,0,,,ASPECT
    CIRCLE(200,60),55,3,,,ASPECT
    ASPECT = ASPECT+1.4
  WEND
GET (0,0)-(327,127),P
CheckMouse:
  IF MOUSE(0)=0 THEN CheckMouse
  IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
  IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture:
  PUT(X,Y),P
  X=MOUSE(1): Y=MOUSE(2)
  PUT(X,Y),P
  GOTO CheckMouse
  
```

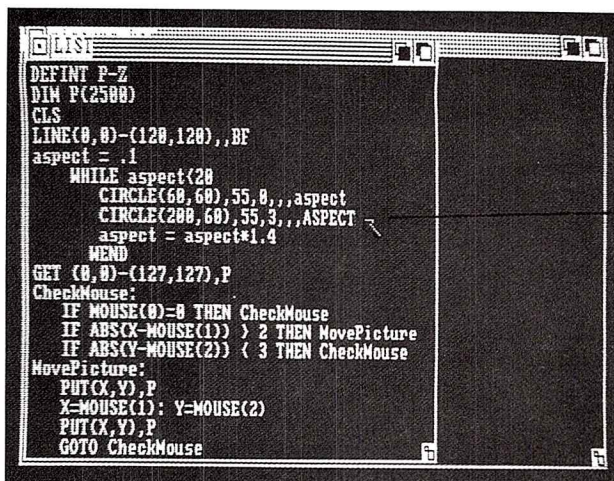
Find line of code that
draws the first sphere

- Click at the end of the line to move the insertion point there.
- Press the RETURN key to open a new line.

Now you are ready to type a new line. Note that BASIC automatically aligns the cursor with the statement directly above it saving you the bother of inserting blank spaces.

- Type the following line:

```
CIRCLE(200,60),55,3,,,ASPECT
```



```
LIST
DEFINT P-Z
DIM P(2500)
CLS
LINE(0,0)-(120,120),,BF
aspect = .1
WHILE aspect<28
  CIRCLE(60,60),55,0,,,aspect
  CIRCLE(200,60),55,3,,,ASPECT
  aspect = aspect*1.4
WEND
GET (0,0)-(127,127),P
CheckMouse:
IF MOUSE(0)=0 THEN CheckMouse
IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture:
PUT(X,Y),P
X=MOUSE(1): Y=MOUSE(2)
PUT(X,Y),P
GOTO CheckMouse
```

Enter this line of code to draw the second sphere

This statement draws an ellipse with the center located at 200,60. It has a radius of 55 and an aspect ratio equal to ASPECT. In Amiga Basic, the number 0 represents blue, and the number 3 represents orange. Every time the WHILE loop is executed, the statement draws another ellipse with a different aspect ratio (ASPECT). These ellipses form the sphere.

- Choose Start to run the program.

Correcting Errors

You may introduce an error when you type or edit a program. When it finds an error, BASIC stops program execution and displays a requester describing the error. BASIC makes sure the List window is visible and then scrolls the window so the line containing the error is visible in the window. The statement that caused the error is enclosed in an orange rectangle. Then you can edit the incorrect line in the List window and run the program again.

Replacing a Program Line

Since you changed the program, only the first sphere moves when you click the Selection button. Let's change the program so that the both spheres move together.

- If the program is still running, choose Stop to stop it.

Choose Show List Window. Observe that Show List Window doesn't change the position of the List window.

- Point at the extreme left edge of the GET statement and drag the highlighting across to the end of the line. Note that this selects the entire line, highlighting it in orange.

```

LIST
DEFINT P-Z
DIM P(2500)
CLS
LINE(0,0)-(120,120),,BF
aspect = .1
WHILE aspect<20
  CIRCLE(60,60),55,0,,,aspect
  CIRCLE(200,60),55,3,,,ASPECT
  aspect = aspect*1.4
WEND
GET(0,0)-(327,127),P
CheckMouse:
IF MOUSE(0)=0 THEN CheckMouse
IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture:
PUT(X,Y),P
X=MOUSE(1): Y=MOUSE(2)
PUT(X,Y),P
GOTO CheckMouse

```

Select the GET statement

- Choose Cut from the Edit menu to delete the selection.
- On the blank line type

```
GET(0,0)-(327,127),P
```

This new GET statement increases the area that moves when you click the Selection button.

Now, let's change the DIM statement to create an array of 6000 rather than 2500 elements.

- Move the insertion point to the DIM statement.
- Select the part of the statement that reads 2500 and select Cut from the Edit menu.
- Type 6000 within the parentheses so that the line now reads

```
DIM P(6000)
```

```
LIST
DEFINT P-Z
DIM P(6000)
CLS
LINE(0,0)-(120,120),,BF
aspect = .1
WHILE aspect<20
  CIRCLE(60,60),55,0,,,aspect
  CIRCLE(200,60),55,3,,,ASPECT
  aspect = aspect*1.4
WEND
GET (0,0)-(327,127),P
CheckMouse:
IF MOUSE(0)=0 THEN CheckMouse
IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture:
PUT(X,Y),P
X=MOUSE(1): Y=MOUSE(2)
PUT(X,Y),P
GOTO CheckMouse
```

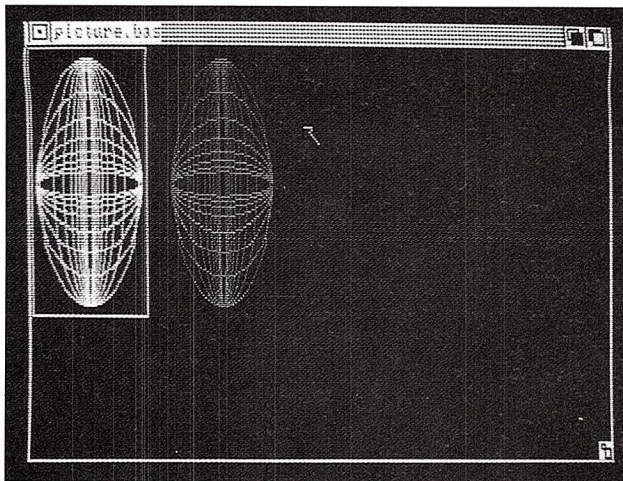
→ Ammended Statements

- Choose Start to run the program.

Now both spheres move together when you click and drag the mouse.

Reversing Blue and White

Let's change the first sphere so that it appears in white on a blue background like this:



- If the program is still running, choose Stop to stop it.
- Find the Line statement in the program.
- Point to the end of the statement and click, putting the insertion point directly after BF.

```

LIST
DEFINT P-Z
DIM P(6000)
CLS
LINE(0,0)-(120,120),,BF
ASPECT = .1
  WHILE ASPECT<20
    CIRCLE(60,60),55,0,,,ASPECT
    CIRCLE(200,60),55,3,,,ASPECT
    ASPECT = ASPECT*1.4
  WEND
GET (0,0)-(327,127),P
CheckMouse:
  IF MOUSE(0)=0 THEN CheckMouse
  IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
  IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture:
  PUT(X,Y),P
  X=MOUSE(1): Y=MOUSE(2)
  PUT(X,Y),P
  GOTO CheckMouse

```

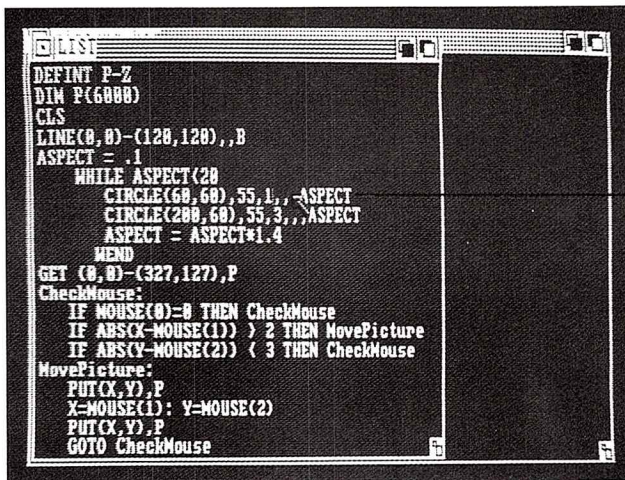
- Press the BACKSPACE key once to delete the F in BF.

Now the inside of the box will be blue, not white.

- Find the line

```
CIRCLE(60,60),55,0
```

- Position the insertion point after the number 0.
- Press the BACKSPACE key once to delete the 0.
- Type 1 to make the color number 1.



```

LIST
DEFINT P-Z
DIM P(6000)
CLS
LINE(0,0)-(120,120),,B
ASPECT = .1
  WHILE ASPECT<20
    CIRCLE(60,60),55,1,-ASPECT
    CIRCLE(200,60),55,3,,ASPECT
    ASPECT = ASPECT*1.4
  WEND
GET (0,0)-(327,127),P
CheckMouse:
  IF MOUSE(0)=0 THEN CheckMouse
  IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
  IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture:
  PUT(X,Y),P
  X=MOUSE(1): Y=MOUSE(2)
  PUT(X,Y),P
  GOTO CheckMouse
  
```

Insert 1

Now the ellipse will be drawn in white instead of blue.

- Choose Start to see the new program output.

The changes in the program are now complete.

Single-Stepping Through The Program

To get better acquainted with Picture, let's use a common debugging technique: single-stepping through the program.

- If Picture is still running, choose Stop to stop it.
- Select the Output Window by clicking anywhere in it. Observe the Ok prompt.
- Type in

end

and press the RETURN key.

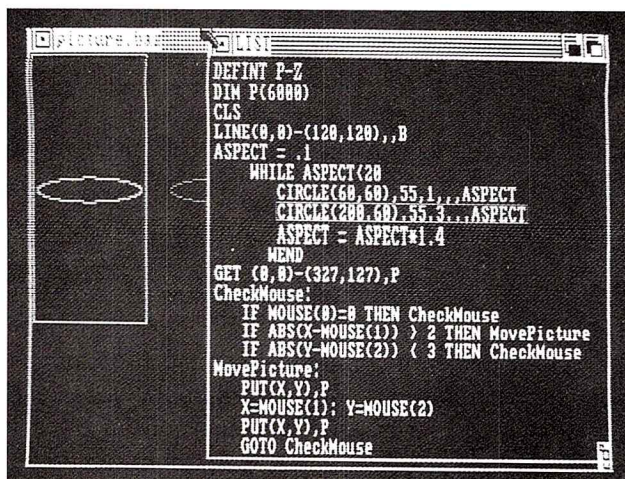
- Choose Step from the Run menu. Step executes the first line of the program and then stops.
- Choose Show List window from the Windows menu to open and select the List window on the right side of the screen.

Each statement is outlined in the List window as it is executed. The Output window is selected so that any text you type appears there.

- Choose Step again (or press right Amiga-T).

The next line executes, and the program stops again. There's no output yet, so not much is happening.

Continue choosing Step and watch the program execute one program statement at a time. When the section that draws the ellipses is outlined, observe how it draws the spheres. Each iteration of the WHILE loop adds an ellipse with a different ASPECT (aspect ratio) to each sphere.



```
DEFINT P-Z
DIM P(6000)
CLS
LINE(0,0)-(120,120),,B
ASPECT = .1
WHILE ASPECT<20
  CIRCLE(60,60),55,1,,,ASPECT
  CIRCLE(200,60),55,3,,,ASPECT
  ASPECT = ASPECT*1.4
WEND
GET (0,0)-(327,127),P
CheckMouse!
IF MOUSE(0)=0 THEN CheckMouse
IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture!
PUT(X,Y),P
X=MOUSE(1): Y=MOUSE(2)
PUT(X,Y),P
GOTO CheckMouse
```

- Just for fun, after the first few ellipses have been drawn, type

print aspect

in the Output window, and

- press the RETURN key.

The current value of ASPECT (the aspect ratio for the ellipse) appears in the Output window.

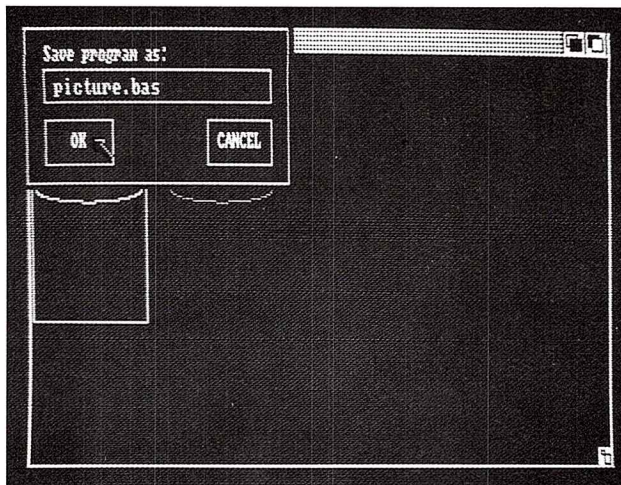
Even though we're not actually debugging Picture, this illustrates a typical "debugging" technique. You can enter a command in the Output window from BASIC "on the spot." This is called entering a command in "immediate mode." BASIC executes immediate mode commands immediately and displays the result if there is one. See "Operating Modes" in Chapter 3, "Using Amiga Basic," for more information on immediate mode.

- Continue stepping through Picture. Check other variables if you like.
- If you'd rather stop stepping and just run the rest of the program, choose Continue from the Run menu.

Saving The Program

Whenever you enter a new program or make changes to an existing program, use the Save As menu item to put the program on the disk. Once it's on the disk, you can load and run it any time you like.

- Choose the Save As item from the Project menu. The following requestor appears:



BASIC assumes you want to save the program under its current name, `picture.bas`. It also assumes that you want to save the program in whatever form it was loaded in.

You can change the name if you want to, but the simplest thing is to simply click the OK Gadget.

Now you have two versions of `picture.bas` on the disk: the original, unchanged `picture2.bas` and the newly edited `picture.bas`. You could have also decided to rename the program as "`myprogram.bas`" or any other legal name. That would have preserved `picture.bas` in the form that you found it before your changes.

Leaving BASIC and Returning to the Workbench

- Choose Quit from the Project menu.

Congratulations! You have just finished the practice session.

You are now back at the Workbench ready to begin your next activity on the Amiga. But you've learned a lot about Amiga Basic in just a few minutes.

You've learned how to

- Load an existing program.
- Edit programs in the List window.
- Work with some BASIC statements and functions.
- Save a BASIC program file.

In the next chapter, you'll find the elementary facts about how to operate BASIC, including a section called "The Amiga Basic Screen." You'll recognize things you discovered in the practice session, and you'll observe new things as well. As with all Amiga tools, you can't "harm" the computer or BASIC through normal typing, mouse pointing, or trial and error. So don't hesitate to experiment with Amiga Basic and try out all the features of the screen.

To load an existing program:

To load an existing program, enter the command:

```
LOAD <filename>
```

To edit the loaded program:

To edit the loaded program or enter a new program, enter the command:

```
LIST [<line number>]
```

LIST calls BASIC's full screen editor. If you specify a line number, that line appears on the top line of the display along with the 24 lines that follow it.

To execute a program in memory:

To start a program in memory running, enter the command:

RUN

To stop the program while it is running, press CTRL-C.

To debug the program, you can use immediate mode statements. For example,

```
FOR I=1 to 20 PRINT A(I): NEXT I
```

To resume execution of the program, enter the following command:

CONT

To leave BASIC:

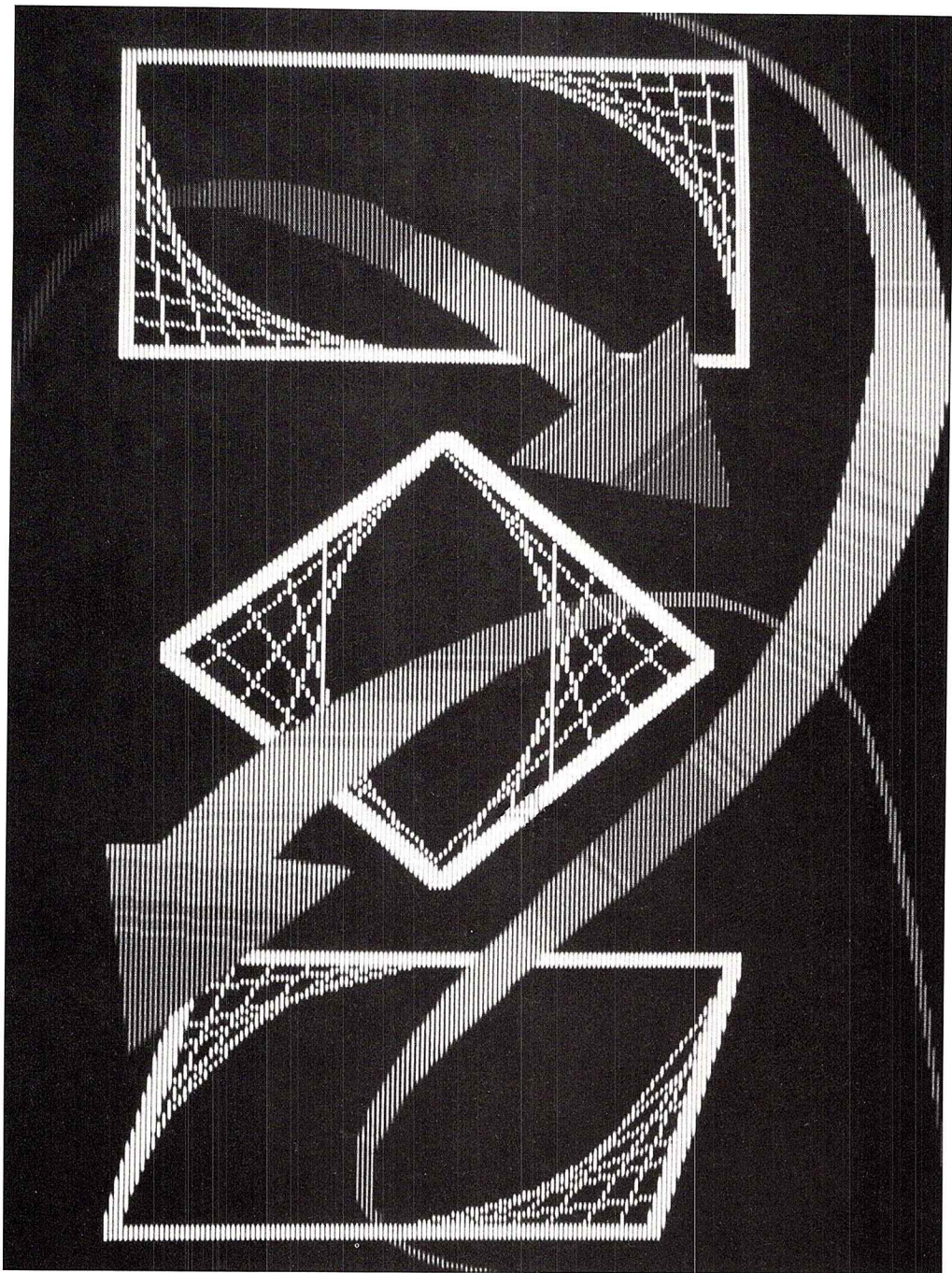
To quit the BASIC and return to the Workbench, enter the command:

SYSTEM

To save a program currently in memory:

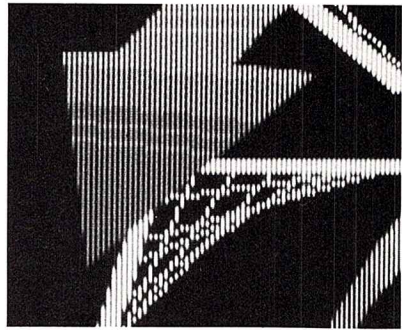
If the program currently in memory has been altered and not saved, the following message appears to prompt you:

```
Program not saved, type SYSTEM again if you don't  
want to save it.
```

Chapter 3

Using Amiga Basic



This chapter contains the fundamental operating information for using Amiga Basic, including how to start and quit BASIC, how to load and save files, and how to use the different operating modes. It then goes on to describe the various elements of the Amiga Basic screen.

Operating Fundamentals

The following section explains how to start and exit Amiga Basic and how to load and save Amiga Basic programs.

Starting Amiga Basic

Here are the two ways to start Amiga Basic:

- Open the Amiga Basic icon on Workbench.
- Type

`basic`

on the CLI screen (selected from Preferences) and press the RETURN key. You can also double-click on any Amiga Basic program icon in the Workbench. Not only does this invoke BASIC, it also loads and runs the selected program.

Exiting Amiga Basic and Returning to the Workbench

There are two ways to exit Amiga Basic and return to the Workbench.

- Select the Quit item from the Menu Bar's Project menu.
- Type

`system`

in the selected Output window and press the RETURN key or enter SYSTEM as an instruction in a BASIC program.

Loading a Program

To run a program, the program must be in memory. There are several ways to put an existing program into memory.

- When in the Workbench, double-click the icon for an Amiga Basic program. This loads BASIC and loads and runs the selected program.
- If BASIC has already been loaded, you can select the Open item from the Project menu. This displays a requester asking you which program you wish to load. Type in the name of the program and click in the OK Gadget (or press the RETURN key).
- If BASIC has already been loaded, you can type the LOAD statement in the Output window. See "LOAD" in Chapter 8 for the proper syntax for this statement.
- If a BASIC program is currently running, it can use the CHAIN statement to load and run another program.

Saving a Program

To save a new program, you can either select the Save As item from the Project Menu or type the SAVE statement in the Output window. See "SAVE" in Chapter 8 for the proper syntax of this statement. To file away a previously saved and now re-edited program, you can either enter the SAVE command or select the Save item from the Project menu (see below).

Amiga Basic saves all new programs in compressed form unless you explicitly instruct it to do otherwise with the SAVE statement. To save programs for a word processor or in protected form, you must also give explicit instructions with the SAVE command in the Output Window.

Operating Modes

When you open Amiga Basic, the Output window appears with the name BASIC. It is ready to accept commands. At this point, you can use Amiga Basic in one of three modes: immediate mode, edit mode, or program execution mode. The List window is selected when BASIC begins operating.

Immediate Mode

In immediate mode, BASIC commands are not stored in memory, but instead are executed as they are entered in the Output window. Results of arithmetic and logical operations are displayed immediately and stored for later use, but the instructions themselves are lost after execution. Immediate mode is useful for debugging and for using BASIC as a calculator for quick computations that do not require a complete program.

To begin entering immediate commands, you must first select the Output window by clicking anywhere in it with the Selection button.

Program Execution Mode

When a program is running, BASIC is in program execution mode. During program execution, you cannot execute commands in immediate mode, nor can you enter new lines in the List window.

Edit Mode

You are in edit mode when you are working in the List window.

The Amiga Basic Screen

There are three separate regions of the BASIC screen: the Output window, the List window, and the Menu Bar.

The Output and List windows have these traits in common;

- To select a window, you click anywhere inside it.
- To resize the window, you drag the Sizing Gadget in the lower right corner.
- To bring the back window to the front, you click the Front Gadget.
- To put the front window to the back, you click the Back Gadget.
- To close the window, you click the Close Gadget located in the upper left corner.
- To move the window, you press and hold down the Selection button and drag the Title Bar. (You can also move the Output window if you resize it.)

The menu bar has several distinguishing qualities:

- To display the Menu Bar, you select the List or Output window and press and hold down the Menu button.
- To display the individual menus, you point at the desired menu title.
- To choose an individual menu item, you point at it to highlight it, and release the Menu Button.

The following sections describe additional features of each of these screen areas.

The Output Window

You can use the Output window both to enter statements as immediate mode commands and to display the output from your programs.

To select the Output window:

- Click inside it, or
- Choose Show Output Window from the Windows menu, and click inside it.

In the Output window, you can:

- Enter a statement as a immediate mode command. BASIC executes the command when you press the RETURN key. Any output from the command appears in the same Output window.
- Use the BACKSPACE key to delete typing mistakes before you correct them.

The List Window

You can use the List window to enter, view, edit, and trace the execution of programs. The List window is automatically selected when you first open Amiga Basic.

To select the List window:

- Click inside it, or
- Choose Show List Window from the Windows menu, and click inside it.

The List window is made visible when the program halts due to an error.

Note: If a program has been saved in a protected file (with the SAVE command in the Output window), you cannot open a List window for the file. Protected files can neither be listed nor edited.

In the List window, you can:

- Look at a program and scroll through it with the arrow keys and the SHIFT and ALT keys combined with the arrow keys.
- Enter or edit a program using all of the features of Amiga Basic, including selecting text with the mouse and using the options in the Edit menu. See "List Window Hints" in Chapter 4, "Editing and Debugging Your Programs," for more details on List windows.

The Menu Bar and Menu Keyboard Shortcuts

There are four menus on the Menu Bar: Project, Edit, Run, and Windows. You cannot always use all of these menus. A menu title may be displayed less distinctively as a *ghost* menu item to indicate that the menu is not relevant to what you are doing at the moment. Similarly, a ghost menu item may appear when a ghost menu or menu item cannot be selected.

Some of the menu items show an Amiga key sequence next to them, such as Amiga-X for Cut. This means you can press the given key combination (press the "X" key while holding down the right Amiga key) instead of choosing the item with the mouse, if you want to. All the menu keyboard shortcuts use the right Amiga key.

The Project Menu

The Project menu contains five items that affect program files. There are no keyboard shortcuts for the items in the Project Menu.

New gets BASIC ready to accept a new program. It clears the current program listing from your screen and clears the program from memory, so you can begin a new program. It behaves the same way as the New statement.

Open tells BASIC that you want to bring in a program that is already on the disk. To display the names of the programs on the disk, select the Output window and enter the FILES command. When you choose Open, a requester appears to ask which program you wish to open. Type in the name of the desired program, then click the OK Gadget.

Save saves the program under its current name. This means it puts a program on the disk after you have entered it or made changes to it. Save saves all new programs in compressed format and saves all revised programs in whatever format they were loaded in.

Save As... is the same as Save, except that Save As allows you to change the name of the program to be saved. Amiga Basic saves your new programs in compressed format, and it saves your loaded and revised programs in whatever form they were loaded in.

To save your program in text or protected format, you must use the Save statement as a immediate mode command in the Output window. See "Program File Commands" in Chapter 5, "Working with Files and Devices," for an explanation of file formats. See SAVE in Chapter 8, "BASIC Reference," for the syntax and grammar of the SAVE statement.

Quit tells Amiga Basic to return to the Workbench. It behaves exactly like the SYSTEM statement.

The Edit Menu

The Edit menu has three items that are used when entering and editing programs. Except for immediate mode commands in the Output window, you enter and edit all program statements in the List window. Each of the Edit menu commands has a keyboard shortcut.

Cut deletes the current selection from the List window and puts it in the Clipboard. Pressing Amiga-X is the same as choosing Cut.

Copy puts a copy of the current selection into the Clipboard without deleting it. Pressing Amiga-C is the same as choosing Copy.

Paste replaces the current selection with the contents of the Clipboard. If no characters are selected, Paste inserts the contents of the Clipboard to the right of the insertion point. Pressing Amiga-P is the same as choosing Paste.

The Run Menu

The Run menu has six commands that control program execution. Keyboard shortcuts are available for four of these commands.

Start runs the current program. Entering RUN in the Output window or pressing Amiga-R are the same as choosing Start. Start is enabled whenever BASIC is in immediate mode. Pressing Amiga-R is the keyboard shortcut for running the current program.

Stop stops the program that is running. Stop behaves exactly like the STOP statement. Amiga-. or CTRL-C are the keyboard shortcuts for stopping the current program.

Continue starts a stopped or suspended program. Entering CONT in the Output window is the same as choosing Continue. The Continue menu item is enabled only when a program has actually been stopped and continuing is possible. If no program was stopped, or if you changed the program while it was stopped, a requester appears that says "Can't continue."

Suspend suspends the program that is running until any key other than Amiga-S is pressed. Pressing Amiga-S or CTRL-S are the same as selecting Suspend. Suspend is enabled whenever a program is running.

Trace On is a toggle that turns program tracing on and off for debugging. If the List window is visible, tracing highlights each statement as it is executed. This works the same as the TRON statement where the last statement executed has a trace rectangle drawn around it. If no statement has been executed, no rectangle is drawn. This lets you determine where the program is being stopped.

Trace Off works the same as the TROFF statement where tracing no longer highlights each statement as it occurs

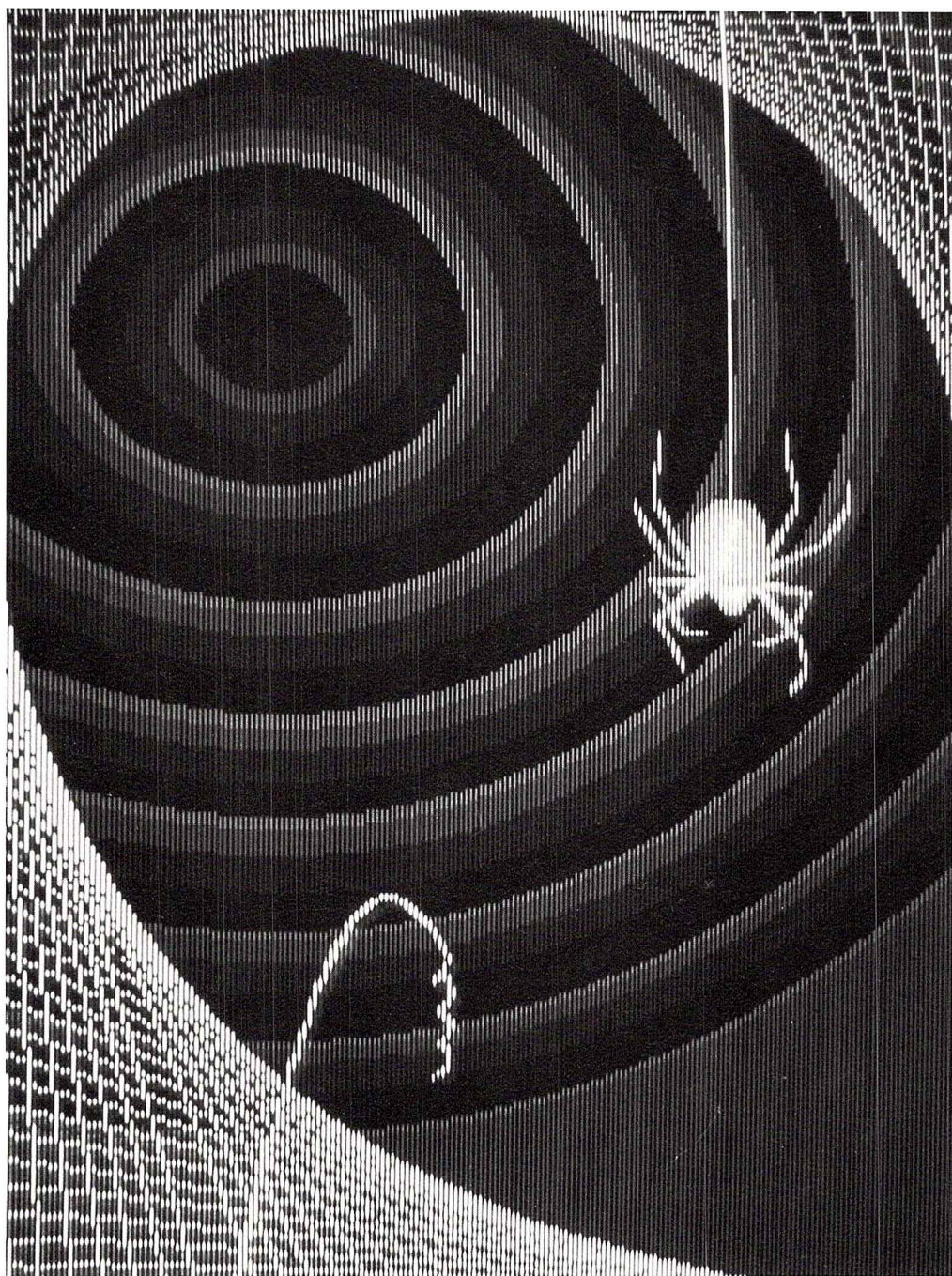
Step executes the program one step at a time. It stops after each statement. Pressing Amiga-T is the same as choosing Step. When the List window is made visible, a rectangular box outlines the statement that was just executed.

The Windows Menu

The Windows menu has two items that open windows on the BASIC screen.

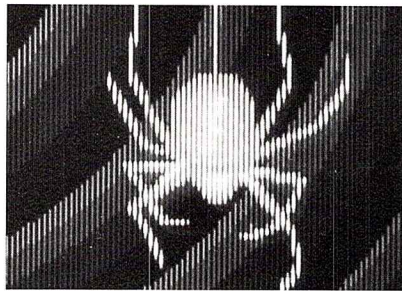
Show List Window opens and selects the List window on the current program. If a List window is already opened but covered with the Output window, Show List Window brings it forward. Pressing Amiga-L is the same as choosing Show List Window. To edit a loaded program or to enter a new program, you can also use the LIST immediate mode command in the Output window.

Show Output Window opens the Output window. The List window is put behind the Output window. In order to enter immediate mode commands in the Output window, you must first click in it.



Chapter 4

Editing and Debugging Your Programs



This chapter describes how to enter text to write programs and how to remove errors from programs.

Editing Programs

The List window appears when you start Amiga Basic. Use the regular Amiga editing commands, Cut, Copy, and Paste, to enter and edit the program lines in the List window.

When you first open BASIC, the List window that appears may seem too narrow to use for long program lines. Text that you enter beyond the right margin forces the window to scroll, keeping the cursor in the visible part of the List window. To get back to the left margin, press ALT-Left Arrow. Drag the List window to the left, and then drag the Sizing Gadget to the right to increase the width of the right margin.

Editing Reminders

Editing program lines in the List window is much like working with regular text on a processor.

Here are some reminders about typing, selecting, and editing text in the List window.

Typing and Editing Text

- Insert text by typing it or by pasting it from the Clipboard. Inserted text appears to the right of the insertion point.
- Delete text by backspacing over it or by selecting it and then choosing Cut from the Edit menu.
- End each program line with a carriage return. You can have extra carriage returns in your BASIC programs. However, these only create blank lines that are ignored when the program executes.
- You can indent lines of text by using the TAB key. The TAB key advances three characters to the right. When you press the RETURN key at the end of a line, the cursor descends one line and goes to the column where the previous line started. This means if the previous line started with a tab, the new line starts at the same tab stop. This indentation does not cost additional memory.

- You can type reserved words in either uppercase or lowercase, but BASIC always displays them in uppercase.
- You can type variable names of up to 40 significant characters. A variable is initially single precision unless you terminate it with a special character or execute a DEFINT, DEFLNG, DEFSNG, DEFDBL, or DEFSTR statement. The special characters are \$ for string, ! for single precision, # for double precision, % for integer, and & for long. You can type variable names in either upper or lowercase, but BASIC does not distinguish between them. For example, alpha, Alpha, and ALPHA all refer to the same variable.
- You can precede program lines with numbers, but line numbers are not required.

Selecting Text

- Select characters or lines by dragging the highlighting over them with the mouse.
- The quickest way to select a single line is to point at the far left edge of the line and drag the highlighting down one line.
- If you drag the highlighting to the edge of the List window and keep holding down the Selection button, the window automatically scrolls, selecting as it goes.
- Select individual words in program lines by pointing at them and double-clicking.

An alternative way to make an extended selection is to click at the beginning of the selection, move to the end of the selection, and Shift-click (click while holding down the SHIFT key). This action selects all the text between the beginning and the end of the selection.

Scrolling

- When you reach the bottom of a List window and continue entering lines, BASIC automatically scrolls up one line at a time.
- BASIC automatically scrolls horizontally when you reach the right edge of a List window and continue typing.
- Use the four arrow keys to move the insertion point one character to the right or left or one line up or down.
- If you press the right arrow key and the insertion point is already at the rightmost column of the display, the display scrolls down 75% to the right. If the display has already scrolled as far to the right as possible, BASIC beeps to indicate it can go no further. The left, up, and down arrows behave in the same way.
- If you hold the SHIFT key down while you hold down any arrow key, the display scrolls in that direction. If it has already scrolled as far as possible in that direction, BASIC beeps.
- To move forward through a program listing a windowful at a time, press SHIFT-Down Arrow. To move backwards through a program listing a windowful at a time, press SHIFT- Up Arrow.
- To move to the beginning of a program listing, press ALT-Up Arrow. To move to the end of a program listing, press ALT-Down Arrow.
- To move to the far right margin of a given program line, press ALT- Right Arrow. To move to the far left margin of a given program line, press ALT- Left Arrow.
- To move 75% of the way towards the right margin of a given program line, press SHIFT-Right Arrow. To move 75% of the way towards the left margin of a given program line, press SHIFT-Left Arrow.

Opening the List Window at a Specific Line or a Specified Label

To open the List window at a specified line, enter the LIST command in the Output window and include a label or a line number. The List window opens with that line as the first line.

For example, LIST MovePicture opens the List window on the picture.bas program beginning with the MovePicture subroutine.

Debugging Programs

This section describes the four debugging features that Amiga Basic provides: error messages, the TRON command, the Step option and the Suspend option. You can use these features to save time and effort while removing program errors.

Error Messages

When a program encounters an error, program execution halts, a requester appears with the error message, and the line with the error is outlined in the List window. See Appendix B, "Error Codes and Error Messages," for a complete listing of these codes and messages with some probable causes and suggestions for recovery.

TRON Command

It is easy to remember the TRON command as TRace ON. You are in Trace mode whenever you choose the Trace On item from the Run menu, execute the TRON statement in a program line, or enter TRON in the Output window.

If the List window is visible, the statement being executed is framed with an orange rectangle. As the program executes, statement by statement, each statement is framed.

To disable TRON, select the Trace Off item from the Run menu, execute TROFF in a program line, or enter TROFF in the Output window.

If you have isolated the error to a small part of the program, it is easier and quicker to turn on TRON from within the program, just before the error is reached.

Step Option

The Step option executes the next statement of the program in memory. If the program has been executed and stopped, Step executes the first statement following the STOP statement. The program then returns to immediate mode. If there is more than one statement on a line, Step executes each statement individually. You can choose the Step item in the Run menu or press Right Amiga-T.

If the List window is visible, Step frames the last statement that has been executed.

You can advance through a program, step by step, testing results at the end of each line, and interactively testing variable values by using the PRINT command in the Output window.

To reset Step to start at the beginning of a program, enter the END statement in the Output window.

If ON BREAK trapping is enabled, you cannot use Step to stop the execution of a program. For more information, see "ON Break" in Chapter 8, "BASIC Reference."

Suspend Option

To create a pause in program execution, you can choose Suspend from the Run menu or press Right Amiga-S. The pause created lasts until you press any key (except Right Amiga-S) or select Continue from the Run menu. Suspend is enabled whenever a program is running.

Continue Option

To resume execution of a program, you can enter the `CONTINUE` command in the Output window or choose Continue from the Run menu.

Using CUT, COPY, and PASTE Commands in List Windows

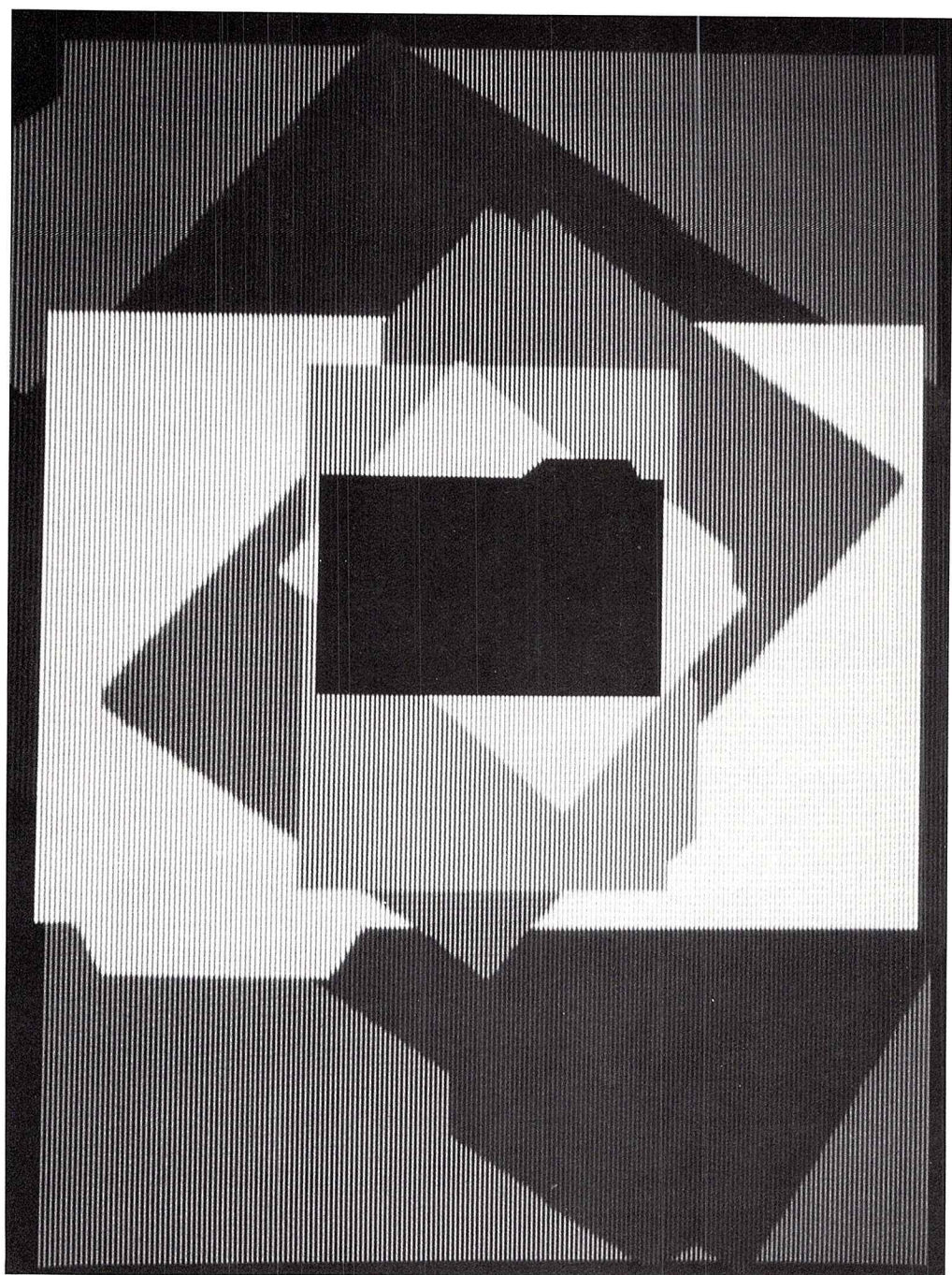
Don't forget that the contents of the Clipboard are replaced with each Cut and Copy command. However, a Paste command does not change the contents of the Clipboard, so you can paste the same contents into different places in a program as many times as you want.

Sometimes you may want to cut something out of the program without having it overwrite information you have on the Clipboard. You can do this by highlighting the text you want to eliminate and pressing the Backspace key. This is also a good technique when you want to avoid generating "Out of heap space" error messages which can occur when deleting a very large block of text.

Using the Output Window for Debugging

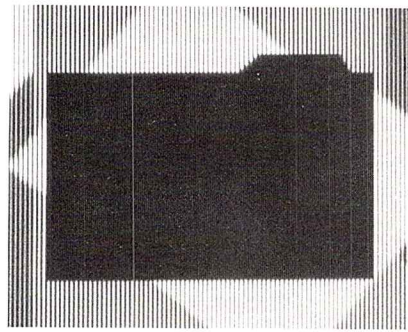
Once a program has been suspended, you can use the Output window to glean useful debugging information in immediate mode. For example, if your program is causing an error message, and the error occurs somewhere within a loop, you can find out how many times the program has executed the loop and all the variable values. You find this out by entering immediate mode instructions in the Output window to `PRINT` the variables (for exact syntax, see "`PRINT`" in Chapter 8, 'BASIC Reference').

Another use of the Output window in debugging is to change the values of variables with immediate mode `LET` statements. You can assign a new value to a variable and use the Continue selection on the Run menu to resume program execution.



Chapter 5

Working with Files and Devices



This chapter discusses the way to input and output information through the system and the way Amiga Basic uses files and drives. In addition, it describes file handling and gives some suggestions for transferring data between Amiga Basic and a word processor.

Generalized Device I/O

Amiga Basic supports generalized input and output. This means that various devices can be used with the same syntax BASIC uses to access disk files. The following devices are supported:

- SCRN:** Files can be opened to the screen device for output. All data opened to SCRn: is directed to the current Output window.
- KYBD:** Files can be opened to the keyboard device for input. All data read from a file opened to KYBD: comes from the Amiga keyboard.
- LPT1:** Files can be opened to this device for output. All data written to a file opened to LPT1: is directed to the line printer. If "LPT1:BIN" is specified, BASIC performs BINARY output to the line printer. Then BASIC does not expand tabs into spaces or force carriage returns when the printer's width is exceeded.
- COM1:** Files can be opened to this device for input or output. Files opened with COM1: communicate with the Amiga serial port. BASIC recognizes the following parameters as part of the "COM1:" filename:

COM1: [baud-rate] [, [parity] [, [data-bits] [, stop-bits]]]

baud rate the speed at which the Amiga communicates. The baud rate is one of the following values: 110, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, or 19200.

parity a technique for detecting transmission errors. The default is E. It is either O (for odd), E (for even), or N (for none).

data-bits the bits in each byte transmitted that are real data and not overhead (parity bits and stop bits). It is either 5, 6, 7, or 8.

stop-bits used to mark the end of the transmitted "byte." When the baud rate is 110, the default for stop-bits is 2. At all other baud rates, the default is 1. When 2 stop bits and 5 data bits are specified, 1.5 stop bits are used. For example,

```
OPEN "COM1:300,N,7,2" AS #1
```

Printer Option

Amiga Basic provides one way to use the printer. You address the printer device, LPT1:.

File Naming Conventions

There are few filename constraints in Amiga Basic on the Amiga. All files have a filename preceded by an optional volume name.

Filenames

Amiga Basic pathnames can be from one to 255 characters in length, and can consist of either uppercase or lowercase alphanumeric characters or a combination of both. Each file or subdirectory name within a path is limited to 30 characters. No Command characters can be used in filenames. Here are some examples of valid filenames:

```
PAYROLL  A2400    MyFile    CHECK REGISTER
```


Volume Specifications

Your Amiga comes with one built-in disk drive. You may connect an additional disk drive to increase your storage capacity. Even on one-drive systems, some people will have more than one volume. In this case, you must explain which volume is to be activated for loading or saving files. You do this by adding the relevant file name to the volume name, separating them by a colon.

For loading program files, it is best to select the Open item on the Project menu. If the program file you wish to load is on another disk, press the eject button next to the built-in disk drive, and insert the disk with the desired file. After the disk is inserted, use the FILES command to display the files on the disk, and you can proceed with selecting and loading the file in the normal way you would if the file was on the same disk. To save program files on another disk, it is best to select the Save As command on the Project menu. The process that follows is similar to the procedures for loading.

You can also load a program from another volume with the LOAD, MERGE, or RUN commands by entering the volume name and filename, separated by a colon, in the Output window. However, if that volume has not been previously mounted on the system, an 'Unknown volume' error message is generated. To avoid this, you will have to first eject the disk in your built-in drive by pressing the eject button. Then you can insert the volume containing the program you wish to load.

Handling Files

This section examines file I/O procedures for the beginning BASIC user. If you are new to Amiga Basic, or if you are encountering file-related errors, read through these procedures and program examples to make sure you are using the file statements correctly.

Program File Commands

The following is a brief overview of the commands and statements you use to manipulate program files. More detailed information and syntactic rules are given in Chapter 8, "BASIC Reference," under the various statement names.

Opening a Program File

There are three main ways to open up a program file. The most common is to use the LOAD command. When you load a program file, all open data files are closed, the contents of memory are cleared, and the loaded program is put into memory.

Another way to get a program file is to bring a program into memory and attach it to the end of a program already in memory. Do this by using the MERGE command. This is useful when you are developing a large program and want to test the parts of it separately. After testing and debugging the parts, you can merge them together.

A third way to get at a program file is to transfer control to it during the execution of another program. Do this by using the CHAIN statement. When you use CHAIN, the program in memory opens up another program and brings it into memory. The first program is no longer in memory. Options to the CHAIN statement permit all or some variable values to be preserved, and merging of the program already in memory with the program to which control is being transferred.

Putting Away Program Files

The two main ways to file away your programs are to select the Save or Save As selections on the Project menu, or to type the SAVE command in the BASIC window. For information on the Save and Save As selections, see "The Menu Bar" in Chapter 3, "Using Amiga Basic." For full details on the SAVE command, see 'SAVE' in Chapter 8, "BASIC Reference." The default format for saved files is binary.

If you wish to have a program protected from being listed or changed, use the "Protected" (,P) option with the SAVE command. You will almost certainly want to save an unprotected copy of a program for listing and editing purposes.

If you wish to save the program in ASCII format, use the 'ASCII' (,A) option. ASCII files use up more room than binary ones, but word processing programs can read ASCII files, and CHAIN MERGE and MERGE can successfully work only with programs in this format.

Additional File Commands

Amiga Basic provides you with additional program file-handling statements as well. The NAME statement provides you with the ability to rename existing program and data files. The KILL statement enables you to delete a data or program file from a volume. For detailed information about these two commands, see "KILL" and "NAME" in Chapter 8, "BASIC Reference."

Data Files – Sequential and Random Access I/O

There are two types of data files that can be created and accessed by a BASIC program: sequential files and random access files.

Sequential Files

Sequential files are easier to create than random access files, but are not as flexible and quick in locating data. The data written to a sequential file is a series of ASCII characters stored, one item after another (sequentially), in the order written. The data is read back sequentially, one item after another.

Warning: Sequential files can be opened in order to write to them or read from them, but not both at the same time. When you need to add to a sequential file that has already been given data and closed, do not open it for output. This erases the previous contents of the file before it writes the new data you give it. Use append mode to add information to the end of an existing file if you don't want to erase existing data.

This version of BASIC gives you the option of specifying the file buffer size for sequential files I/O. The default length is 128 bytes. This size can be specified in the OPEN statement for the sequential file. The sizes you specify are independent of the length of any records you are reading from or writing to the file; they only specify the buffer size. Larger buffer sizes speed I/O operations, but take memory away from BASIC. Smaller buffer sizes conserve memory, but produce lower I/O speed.

The following statements and functions are used with sequential data files:

CLOSE	LOF
EOF	OPEN
INPUT#	PRINT#
INPUT\$	PRINT
USING#	LINE INPUT#
WIDTH	WRITE#
LOC	

Creating a Sequential Data File

Program 1 is a short program that creates a sequential file, "DATA", from information you enter at the keyboard.

Program 1—Creating a Sequential Data File

```
OPEN "DATA" FOR OUTPUT AS #1
ENTER:
    INPUT "NAME ('DONE' TO QUIT)";N$
    IF N$="DONE" THEN GOTO FINISH
    INPUT "DEPARTMENT"; DEPT$
    INPUT "DATE HIRED"; HIREDATES$
    WRITE #1,N$,DEPT$,HIREDATES$
    PRINT
GOTO ENTER
FINISH:
    CLOSE #1
END
```

As illustrated in Program 1, the following program steps are required to create a sequential file and to gain access to the data in it:

1. Open the file in output (to the file) mode.
2. Write data to the file using the WRITE# statement.
3. After you have put all the data in the file, close the file.

A program can write formatted data to the file with the PRINT # USING statement. For example, the statement

```
PRINT#1, USING"####.##, ";A,B,C,D
```

can be used to write numeric data to the file with commas separating the variables. The comma at the end of the using string in PRINT # USING statements separates the items in the file with commas. It is good programming practice to use “delimiters” of some kind to separate different items in a file.

If you want commas to appear in the file as delimiters between variables without having to specify each one, the WRITE # statement can also be used. For example, you can use the statement

```
WRITE #1,A,B$
```

to write these two variables to the file with commas delimiting them.

Reading Data from a Sequential File

Now let's look at Program 2. It gains access to the file "DATA" that was created in Program 1 and displays the names of employees hired in 1981.

Program 2--Accessing a Sequential Data File

```
OPEN "I",#1,"DATA"  
WHILE NOT EOF(1)  
    INPUT #1,N$,DEPT$,HIREDATE$  
    IF RIGHT$(HIREDATE$,2)="81" THEN PRINT N$  
WEND
```

Program 2 reads each item in the file sequentially and prints the names of employees hired in 1981. When all the data has been read, the WHILE WEND control structure uses the EOF function to test for the end-of-file condition and avoids the error of trying to read past the end of the file.

Adding Data to a Sequential Data File

If you have a sequential file on the disk and want to add more data to the end, you cannot simply open the file in output mode and start writing data. As soon as you open a sequential file in output mode, you destroy its current contents.

Instead, use append mode. If the file doesn't already exist the OPEN statement works exactly as it would if the output mode had been specified.

The following procedure can be used to add data to an existing file called "FOLKS."

Program 3-Adding Data to a Sequential Data File

```
OPEN "A",#1,"FOLKS"
REM***Add new entries
NEWENTRY:
  INPUT "NAME";N$
  IF N$ = "" THEN GOTO FINISH 'Carriage Return exits loop
  LINE INPUT "ADDRESS ? ",ADDR$
  LINE INPUT "BIRTHDAY ? ",BIRTHDATE$
  PRINT #1, N$
  PRINT #1, ADDR$
  PRINT #1, BIRTHDATE$
  GOTO NEWENTRY
FINISH:
  CLOSE #1
END
```

The LINE INPUT statement is used for getting ADDR\$ because it allows you to enter delimiter characters (commas and quotes).

Random Access Files

Creating and accessing random access files requires more program steps than creating and accessing sequential files. However, there are advantages to using random access files. One advantage is that random access files require less room on the disk, since BASIC stores them in a packed binary format. (A sequential file is stored as a series of ASCII characters.)

The biggest advantage to using random access files is that data can be accessed randomly, that is, anywhere in the file. It is not necessary to read through all the information from the beginning of the file, as with sequential

files. This is possible because the information is stored and accessed in distinct units called records. Each record is numbered.

The statements and functions that are used with random access files are:

CLOSE	LOC	OPEN
CVD	LOF	PUT
CVI	LSET	RSET
CVL	MKD\$	
CVS	MKI\$	
FIELD	MKL\$	
GET	MKS\$	

Creating a Random Access Data File

Program 4—Creating a Random Data File

```
OPEN "R",#1,"DATA",32
FIELD #1,20 AS N$,4 AS A$,8 AS P$
START:
  INPUT "2-DIGIT CODE (ENTER -1 TO QUIT)";CODE%
  IF CODE%=-1 THEN QUITFILE
  INPUT "NAME";PERSON$
  INPUT "AMOUNT";AMOUNT
  INPUT "PHONE";TELEPHONE$
  PRINT
  LSET N$ =PERSON$
  LSET A$ = MKS$(AMOUNT)
  LSET P$ = TELEPHONE$
  PUT #1,CODE%
GOTO START
QUITFILE:
CLOSE #1
```

As illustrated by program 4, you need to follow these program steps to create a random access file:

1. OPEN the file for random access. The absence of an input, output, or append parameter specifies a random file. If the record length (LEN=) is not specified, the default value is 128 bytes.

2. Use the FIELD statement to allocate space in a random buffer for the variables to be written to the random access file. The random buffer is an area of memory, a holding area, reserved for transferring data from files to program variables and vice versa.

Here is an example of using the FIELD statement to create a random access file:

```
FIELD #1,20 AS N$, 4 AS ADDR$, 8 AS P$
```

3. To move the data into the random access buffer, use LSET. Numeric values must be made into strings when placed in the buffer. To make these values into strings, use the "make" functions: MKI\$ to make an integer value into a string or MKS\$ to make a single precision value into a string to be stored in a random file.

Here is an example of moving data into the random access buffer:

```
LSET N$ -X$  
LSET ADDR$=MKS$(AMT)  
LSET P$ = TEL$
```

4. To write the data from the buffer to the disk, use the PUT statement and specify the record number with an expression, for example:

```
PUT #1, CODE%
```

Program 4 takes information that is input from the keyboard and writes it to a random access file. Each time the PUT statement is executed, a record is written to the file. The two-digit code that is input in line 30 becomes the record number.

Note: Do not use a fielded string variable in an INPUT or LET statement. BASIC will then redeclare the variable and will no longer associate that variable with the file buffer, but with the new program variable instead.

Accessing a Random Access Data File

Program 5 gains access to the random access file, "DATA", that was created in program 4. When you enter a two-digit code at the keyboard, BASIC reads and displays the information associated with that code from the file.

Program 5—Accessing a Random Data File

```
OPEN "R",#1,"DATA",32
FIELD #1,20 AS N$,4 AS A$,8 AS P$
START:
  INPUT "2-DIGIT CODE (ENTER -1 TO QUIT)";CODE%
  IF CODE%=-1 THEN QUITFILE
  GET #1,CODE%
  PRINT N$
  PRINT USING "$$####.##";CVS(A$)
  PRINT P$: PRINT
  GOTO START
QUITFILE:
  CLOSE #1
```

Follow these program steps to access a random access file:

1. OPEN the file in random mode.
2. To allocate the space in the random access buffer for the variables to be read from the file, use the FIELD statement. (For details on this procedure, see the FIELD statement in program 5, directly above.)

Note: In a program that performs both input and output on the same random access file, you can often use just one OPEN statement and one FIELD statement.

3. To move the desired record into the random access buffer, use the GET statement.

The program can now access the data in the buffer. Numeric values that were converted to strings by the MKI\$ and MKS\$ functions must be converted back to numbers using the "convert" functions: CVI for integers and CVS for single precision values. The MKI\$ and CVI processes mirror each other. MKI\$ converts a number into a format for storage in random files and CVI converts the random file storage into a format that the program can use.

When used with random access files, the LOC function returns the "current record number." The current record number is the last record number that was used in a GET or PUT statement. For example, the following statement:

```
IF LOC(1) > 50 THEN END
```

ends the program execution if the current record number in file #1 is greater than 50.

Random File Operations

Program 6 is an inventory program that illustrates random file access.

Program 6 - Inventory

```
OPEN "INVEN.DAT" AS #1 LEN=39
FIELD #1,1 AS F$,30 AS D$, 2 AS Q$, 2 AS R$, 4 AS P$
FunctionLabel:
CLS:PRINT "Functions:":PRINT
PRINT "1. Initialize file"
PRINT "2. Create a new entry"
PRINT "3. Display inventory for one part"
PRINT "4. Add to stock"
PRINT "5. Subtract from stock"
PRINT "6. Display all items below reorder level"
PRINT "7. Done with this program"
PRINT:PRINT:INPUT "Function";FUNCT
IF (FUNCT>0) AND (FUNCT<8) THEN GOTO Start
GOTO FunctionLabel
Start:
ON FUNCT GOSUB 600,100,200,300,400,500,700
IF FUNCT<7 THEN GOTO FunctionLabel
END
100 :
    GOSUB part
    IF ASC(F$)<>255 THEN INPUT "Overwrite";confirm$
    IF ASC(F$)<>255 AND UCASE$(confirm$)<>"Y" THEN RETURN
    LSET F$=CHR$(0)
    INPUT "Description ";description$
    LSET D$=description$
    INPUT "Quantity in stock ";Quantity%
    LSET Q$=MKI$(Quantity%)
    INPUT "Reorder Level ";reorder%
    LSET R$=MKI$(reorder%)
    INPUT "Unit price ";price
    LSET P$=MKS$(price)
    PUT #1,part%
    INPUT "Press RETURN to continue",DUM$
    RETURN
200 :
    GOSUB part
    IF ASC(F$)=255 THEN GOSUB NullEntry:RETURN
    PRINT USING "Part Number ###";part%
    PRINT D$
    PRINT USING "Quantity on hand #####";CVI(Q$)
    PRINT USING "Reorder level #####";CVI(R$)
    PRINT USING "Unit price $$$$.##";CVS(P$)
    INPUT "Press RETURN to continue",DUM$
    RETURN
300 :
    GOSUB part
    IF ASC(F$)=255 THEN GOSUB NullEntry:RETURN
    PRINT D$
```



```

PRINT "Current quantity: ";CVI(Q$)
INPUT "Quantity to add";additional%
Q%=CVI(Q$)+additional%
LSET Q$=MKI$(Q%)
PUT #1,part%
RETURN
400 :
GOSUB part
IF ASC(F$)=255 THEN GOSUB NullEntry:RETURN
PRINT D$
425 :
INPUT "Quantity to subtract";less%
Q%=CVI(Q$)
IF (Q%-less%)<0 THEN PRINT "Only ";Q%;" in stock":GOTO 425
Q%=Q%-less%
IF Q%<=CVI(R$) THEN PRINT "Quantity now ";Q%
LSET Q$=MKI$(Q%)
PUT #1,part%
INPUT "Press RETURN to continue",DUM$
RETURN
500 :
reorder=0
FOR I=1 TO 100
GET #1,I
IF ASC(F$)=255 GOTO 525
IF CVI(Q$)<CVI(R$) THEN PRINT D$;" Quantity
";CVI(Q$);TAB(30)
IF CVI(Q$)<CVI(R$) THEN PRINT "Reorder level ";CVI(R$)
IF CVI(Q$)<CVI(R$) THEN reorder=(-1)
525 :
NEXT I
IF reorder=0 THEN PRINT "All items well-stocked."
INPUT "Press RETURN to continue",DUM$
RETURN
600 :
INPUT "Are you sure";confirm$
IF confirm$<>"y" AND confirm$<>"Y" THEN RETURN
LSET F$=CHR$(255)
FOR I=1 TO 100
PUT #1,I
NEXT I
RETURN
part:
Enterno:
INPUT "Part number? ",part%
IF (part%<1) OR (part%>100) THEN PRINT "Bad part number"
IF (part%<1) OR (part%>100) THEN GOTO Enterno
GET #1,part%
RETURN
NullEntry:

```

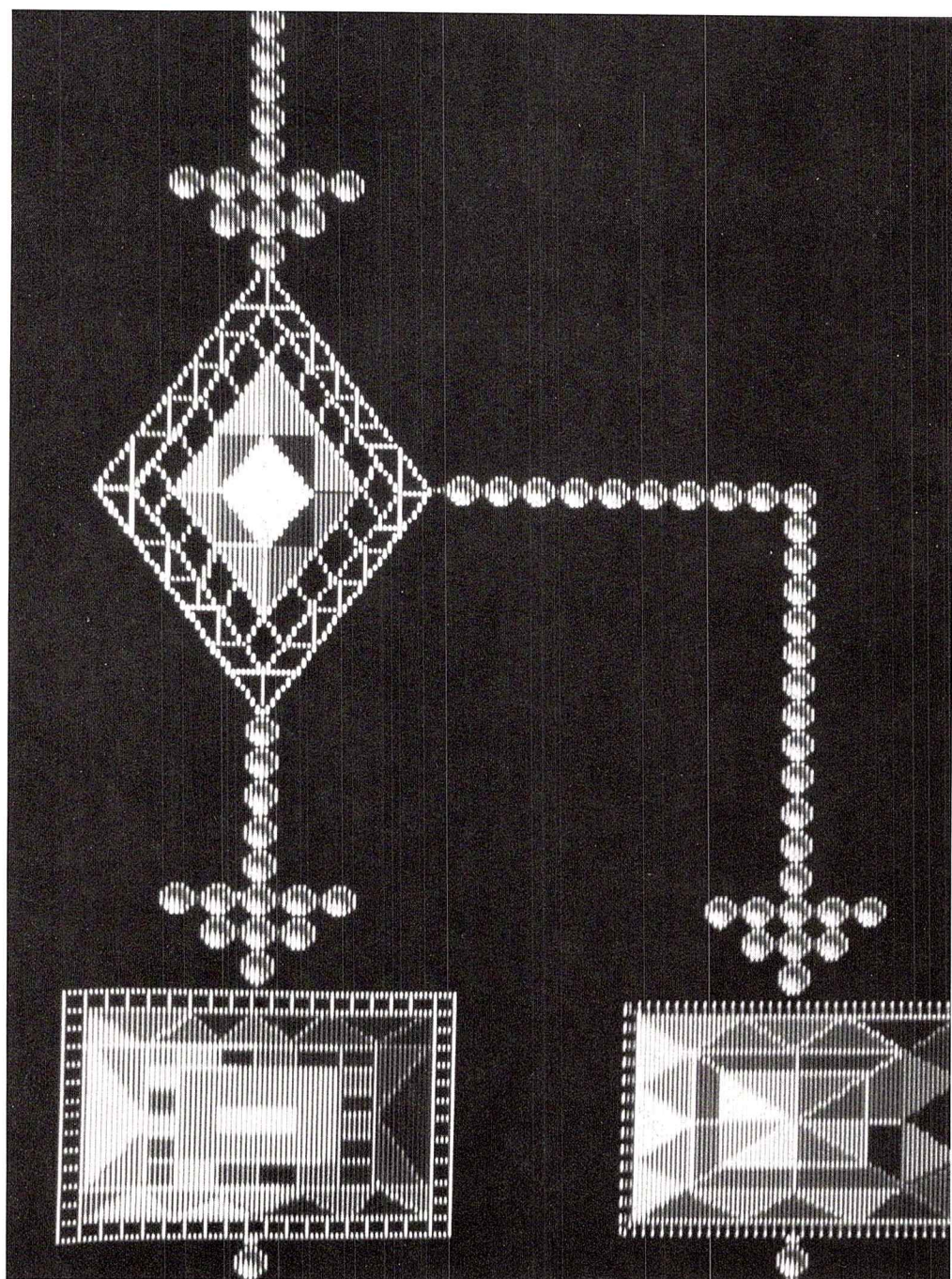
```
PRINT "Null Entry."  
INPUT "Please press RETURN",DUM$  
RETURN  
700 : CLOSE #1  
RETURN
```

Transferring Data Between BASIC and a Word Processor

Remember that word processing programs produce files with more characters than the visible ones in your text. Many word processors use special hidden characters to control appearance and format and to control the printer. These characters can ruin your program file.

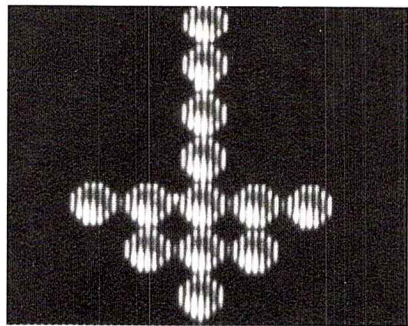
Most, but not all, word processing programs have a filing option called "text only" or "unformatted" or "non-document." When text is filed with this option, all the hidden control characters are removed. Only the text is filed.

Also, if you write a program in Amiga Basic and later wish to use a word processor to edit it, prepare the program first. When you save the BASIC program, use the ",A" (ASCII) option in the SAVE statement which saves the program in a format that can be read by the word processing program.



Chapter 6

Advanced Topics



Amiga Basic supports several advanced programming features including subprograms, event trapping, and memory management. These powerful features, not necessary for beginners to master, add flexibility to Amiga Basic. They are especially helpful to programmers who develop programs for other users.

Subprograms are modules similar to subroutines but with major advantages. They are especially helpful to programmers who write routines that are reused in other programs.

Event trapping allows a program to transfer control to a specific program line when certain events occur. These events include passage of time, mouse activity, a user's attempting to stop the program, or menu selection.

Memory management in Amiga Basic is available through use of the CLEAR statement and the FRE function. These tools can help you create large programs that would ordinarily not run because of the Amiga's limited memory.

Subprograms

Subprograms are sets of program statements similar to subroutines. There are three notable advantages to using subprograms.

First, subprograms use variables that are isolated from the rest of the program. If a programmer accidentally uses a variable name in a subprogram that has already been used in the main program, the two variables still retain separate values. Variables within subprograms are called local variables because their values cannot be changed by actions outside the subprogram.

The second advantage of subprograms is also related to local variables. Programmers frequently find themselves producing the same routine over and over in different programs, rewriting it each time to fit the variable names and design of each new program. Because you don't need to rewrite a subprogram to include it in another program, it is simple to produce a collection of subprograms. Subprograms then can be merged into new programs with minimal changes.

The third advantage of subprograms is that they cannot be executed accidentally. A subroutine can be executed accidentally if no GOTO statement is stationed above it; program flow simply enters the subroutine.

Subprograms do not execute unless a specific CALL to the subprogram is executed.

Referencing Subprograms

Subprograms are referenced by the optional CALL statement with an argument list. (See "CALL" in Chapter 8, "BASIC Reference," for more information.)

In this discussion, you will find references to "formal parameters" and "arguments." Arguments refer to the program variables that are passed in the CALL statement. For example:

```
CALL FIGURETAX(SUBTOTAL, TAX, TOTAL())
```

In this example, the arguments are the variables SUBTOTAL and TAX, and the array variable TOTAL.

Formal parameters refer to the parallel values that the subprogram uses. If, for example, the FIGURETAX subprogram was called using the above CALL statement, the subprogram's first line could appear as:

```
SUB FIGURETAX(FIGURE, TAXRATE, SUM(1)) STATIC
```

In this example, the formal parameters are the variables FIGURE and TAXRATE, and the array SUM. These parameters correspond to (and return values to) the main program variables used as arguments: SUBTOTAL, TAX, and TOTAL().

The parameters that transfer between the main body of the program and the subprogram are said to be passed by reference. This means if the formal parameter is modified by the subprogram, the argument's value changes also.

This can affect the values of variables. For example:

```
CALL AddIt(A,B,C)
.
.
.
SUB AddIt(X,Y,Z) STATIC
  Z = X + Y
```

```

X = X + 12
Y = Y + 94
END SUB

```

If the values of the variables when the program executes the CALL statement are A = 2 and B = 3, then when control returns to the main program, A and B would have altered values. The A variable is tied to X, and B to Y. If the value of X is changed in the subprogram, the value of A is altered as well. In this example, the value of A is increased by 12 in the statement X = X + 12. This subtle change happened because the variable X is an "alias" for the variable A.

In the cases where you want the main program variable's value to change in the subprogram, this works well. Where you don't want this to happen, put parentheses around the variables and they'll retain their values, regardless of what happens in the subprogram. For example:

```
CALL AddIt((A), (B), Result)
```

The parentheses around the first two parameters force them into the category of expressions. Their values cannot be changed by subprograms. You need not use parentheses to pass expressions. For example:

```
CALL AddIt(1+2, 3*A, Result)
```

Note that the type of arguments must match the type of the formal parameters or a type mismatch error will result. For example,

```
CALL DoIt(1)
SUB DoIt (x) STATIC
```

would fail because it tries to pass the integer 1 to the single-precision parameter x.

```
CALL DoIt(1.0)
SUB DoIt(x) STATIC
```

would avoid this error.

Subprogram Delimiters: The SUB and END SUB Statements

Subprograms are delimited by the SUB and END SUB statements. The EXIT SUB statement also can be used to exit a particular subprogram before it reaches the END SUB statement. Execution of an EXIT SUB or END SUB statement transfers program control back to the calling routine.

The syntax is as follows:

```
SUB  subprogram-name [ (formal-parameter-list) ] STATIC
    [ SHARED list-of-variables ]
    .
    .
    .
END SUB
```

The *subprogram-name* can be any valid identifier up to 40 characters in length. This name cannot appear in any other SUB statement.

The *formal-parameter-list* can contain two types of entries: simple variables and array variables. If you are planning to use array variables, read “Declaring Array Parameters” below. Entries are separated by commas. The number of parameters is limited only by the number of characters that can fit on a BASIC line.

STATIC indicates that all the variables within the subprogram retain their values between invocations of the subprogram. Static variable values cannot be changed by actions taken outside the subprogram. STATIC requires that the subprogram be non-recursive; that is, it does not contain an instruction that calls itself or that calls a subprogram that in turn calls the original subprogram.

SHARED variables can be altered by parts of the program outside the subprogram. Those variables you want shared must be explicitly listed in the *list-of-variables* following the SHARED statement. Any simple variables or arrays referenced in the subprogram are considered local unless they have been explicitly declared SHARED variables. See “SHARED” in Chapter 8, “BASIC Reference,” for a discussion of the SHARED statement.

The statements that make up the body of a subprogram are enclosed by the SUB and END SUB statements.

All BASIC statements can be used within a subprogram, except the following:

- User-defined function definitions.
- A SUB/END SUB block. This means subprograms cannot be nested.
- COMMON statements
- CLEAR statements

Declaring Array Parameters

Simple variable parameters can be given any valid Amiga Basic name. Arrays must be declared as follows

array-name ([*number-of-dimensions*])

where *array-name* is any valid Amiga Basic name for a variable and the optional *number-of-dimensions* is an integer constant indicating the number of dimensions in the array. Note that the actual dimensions are not given here.

For example, in the following subprogram,

```
SUB  MATADD2(N%,M%,A(2),B(2),C(3))  STATIC
      .
      .
      .
END SUB
```

N% and M% are integer variables, and A and B are indicated as two-dimensional arrays, while C is a three-dimensional array.

Simple Variables and Array Elements

When a simple variable or array element or an entire array is passed to a BASIC subprogram, it is passed by reference. The following example shows how a subprogram is invoked by the CALL statement, and illustrates call-by-reference argument passing:

```
A = 5 : B = 2
CALL SQUARE(A,B)
PRINT A,B
END

SUB SQUARE(X,Y) STATIC
  Y = X*X
END SUB
```

This example prints the results 5 and 25. Each reference to Y in Subprogram SQUARE actually resulted in a reference to B, and each reference to X resulted in a reference to A. In other words, each time SQUARE used Y, it was actually using B.

Argument Expressions

Expressions also can be passed as arguments to BASIC subprograms. An argument expression is considered to be any valid BASIC expression, except simple variables and array element references. When an expression is encountered in the argument list in a CALL statement, it is assigned to a temporary variable of the same type. This variable is then passed by reference to the subprogram. This is equivalent in effect to the call-by-value passing in functions, whereby the value itself is passed.

If a simple variable or array element is enclosed in parentheses, it is passed the same way as an expression (that is, as call-by-value). For example, if the CALL SQUARE statement in the above example were changed to

```
CALL SQUARE (A, (B) )
```

the results printed would be 5 and 2. In this case (B) is passed by value as an expression, and therefore the subprogram cannot change the value of B.

Note: Arrays should not be passed as parameters to assembly language procedures using the conventions outlined. Instead, the base element of an array should be passed by reference if the entire array needs to be accessed in the assembly language program. For example:

```
CALL X(VARPTR (A(0,0)))
```

Shared and Static Variables in Subprograms

Variables and arrays referenced or declared in subprograms are generally considered to be local to the subprogram. However, Amiga Basic supports shared variables within a module and provides a way for values to be preserved across subprogram invocations.

Shared Variables

By using the SHARED statement in a subprogram, you can access variables without passing them into a subprogram as parameters.

Within a subprogram, main program variables can be used by including the SHARED statement. The SHARED statement only affects variables within that subprogram.

For example:

```
LET A=1: LET B=5: LET C=10
```

```
DIM P(100),Q(100)
```

```
.  
. .  
. . .
```

```

SUB AMIGA STATIC

    SHARED A,B,P(),Q()

    .
    .
    .

END SUB

```

In this example, all main program variables and arrays except C are shared with the subprogram AMIGA.

Static Variables

As already noted, variables and arrays referenced or declared in a subprogram are considered local to the given subprogram. They are not changed by statements outside of the subprogram. Initial values of zero or null string are assumed.

If the subprogram is exited and then reentered, however, variable and array values are those present when the subprogram was exited.

The **STATIC** keyword is required for all subprogram definitions in Amiga Basic.

Array Bound Functions

The upper and lower bounds of the dimensions of an array can be determined by using the functions, **LBOUND** and **UBOUND**.

LBOUND returns the lower bound, either 0 or 1, depending on the setting of the **OPTION BASE** statement. The default lower bound is 0. **UBOUND** returns the upper bound of the specified dimension.

Each function has two syntaxes: a general syntax and a shortened syntax that can be used for one-dimensional arrays. The syntaxes are as follows:

<code>LBOUND (array)</code>	for 1-dimensional arrays
<code>LBOUND (array, dim)</code>	for n-dimensional arrays
<code>UBOUND (array)</code>	for 1-dimensional arrays
<code>UBOUND (array, dim) </code>	for n-dimensional arrays

The *array* is a valid BASIC identifier and the *dim* argument is an integer constant from 1 to the number of dimensions of the specified array.

`LBOUND` and `UBOUND` are particularly useful for determining the size of an array passed to a subprogram.

See “`LBOUND`” in Chapter 8, “BASIC Reference,” for examples of the use of array bound functions.

Event Trapping

Event trapping is a programming capability through which a program can detect and respond to certain “events” and branch to an appropriate routine. The events that can be trapped are time passage (`ON TIMER`), the user attempting to halt the program (`ON BREAK`), the selection of a custom menu item (`ON MENU`), or mouse activity (`ON MOUSE`). BASIC checks between each statement it executes to see if the specified events have happened.

To use event trapping, the programmer builds a subroutine to respond to the event. Then, if the program has activated event trapping for the event, program control is automatically routed to the event-handling subroutine when the event occurs. BASIC does this exactly as if a `GOSUB` statement had been executed to the event-handling subroutine.

The subroutine, after servicing the event, executes a `RETURN` statement. This causes the program to resume execution at the statement that immediately follows the last statement executed before the event trap occurred.

This section gives an overview of event trapping. For more details on individual statements, see Chapter 8, “BASIC Reference.”

Event trapping is controlled by the following statements:

<i>eventspecifier</i> ON	to turn on trapping
<i>eventspecifier</i> OFF	to turn off trapping
<i>eventspecifier</i> STOP	to temporarily turn off trapping

The *eventspecifier* must be one of the following:

TIMER	The timer is the Amiga's internal clock. If you use timer event trapping, you can force an event trap every time a given number of seconds elapses.
MOUSE	Mouse event trapping allows the programmer to redirect program flow when the mouse is clicked by the user.
MENU	If menu event trapping has been activated, the program can use selection of custom menu items as events to trap.
BREAK	When break event trapping is activated, the program sends control to a specified subroutine when the user presses Right Amiga-period, the break keystroke. Care should be taken when using break event trapping. If a programmer uses the statement in a program being tested, the program cannot be exited before a program END statement without rebooting the Amiga. One way to avoid this potential problem is to omit the BREAK ON statement that activates the ON BREAK event trap until testing is completed.
COLLISION	This routine is invoked whenever an object created by the OBJECT.SHAPE statement collides with another object or window border.

ON...GOSUB Statement

The ON GOSUB statement tells BASIC the starting line of the event-handling subroutine. The format is:

ON *eventspecifier* GOSUB *line*

A *line* of zero disables trapping for that event.

Activating Event Trapping

When an *eventspecifier* is ON and if a non-zero line number has been specified in the ON GOSUB statement, each time Amiga Basic starts a new statement it checks to see if the specified event has occurred.

An event will not be trapped by the ON *eventspecifier* statement unless the corresponding *eventspecifier* ON statement has been previously executed.

Terminating Event Trapping

When the *eventspecifier* is OFF, no trapping takes place, and the event is not remembered if it takes place.

Suspending Event Trapping

When the *eventspecifier* is stopped, no trapping takes place. However, the occurrence of an event is remembered so that an immediate trap takes place when an *eventspecifier* ON statement is executed, if the specified event has occurred while the *eventspecifier* was stopped.

When a trap is made for a particular event, the trap automatically causes a STOP on that *eventspecifier*, so recursive traps can never occur. A return from the trap routine automatically reenables the event trap unless an explicit OFF has been performed inside the trap routine.

Note: Once an error trap takes place, all trapping of that event is automatically disabled until a RESUME statement is executed.

Memory Management

Amiga Basic includes the `CLEAR` statement to help writers of large programs manage memory allocation for different purposes.

Using the `CLEAR` statement, you can control the size of three different areas of memory:

- The stack
- BASIC's data segment
- The heap

The Stack

The stack keeps "bookmarks" telling where to return to from `GOSUBS`, nested subprogram calls, nested `FOR...NEXT` loops, nested `WHILE/WEND` loops, and nested user-defined functions.

Conserving Stack Space

Certain Amiga ROM calls require a considerable amount of stack space. The more levels of nesting in your control structures, the more stack space is required to execute a program.

BASIC's Data Segment

BASIC's data segment holds the text of the program currently in memory. It also contains numeric variables and strings. In addition, the data segment contains file buffers for opened files.

Conserving Data Segment Space

A sequential file buffer has a default size of 128 bytes. If your program is tight for memory, one memory reclamation technique is to define a smaller sequential file buffer. A smaller buffer may slow execution of an I/O intensive program, however. See "OPEN" in Chapter 7, "BASIC Reference," for details on changing a sequential file's buffer size. Additionally, the kind of numeric variables you use will have an effect on data segment space. Integer variables take half the number of bytes of single precision; single precision take half the number of bytes of double precision. Also, chaining several small programs together uses less memory than loading and running a large program that incorporates all the smaller ones.

The System Heap

The system heap contains the buffer for SOUND and WAVE information, which, when created, uses 1024 bytes of RAM. The LIBRARY, WINDOW, and SCREEN statements also consume memory from the heap. Amiga Basic shares the System Heap with other tasks running on the Amiga.

Conserving Heap Space

Heap space can be kept smaller by releasing the SOUND/WAVE buffer with a WAVE 0 statement when it is no longer needed.

Using the CLEAR Statement for Memory Management

You can use the CLEAR statement to allocate memory to three areas of RAM.

The syntax of the CLEAR statement is:

```
CLEAR [, [data-segment-size] [, [stack-size]]
```

The *data-segment-size* argument dictates how many bytes are to be reserved for BASIC's data segment.

The *stack-size* argument dictates how many bytes are to be reserved for the stack.

The amount of RAM remaining (Total - (*data-segment* + *stack size*)) is the RAM reserved for the heap. Using the CLEAR statement, your program can define the space it requires for the three adjustable areas of RAM. You can use the FRE functions to find out how much free memory you have in parts of RAM.

Using the FRE Function for Memory Management

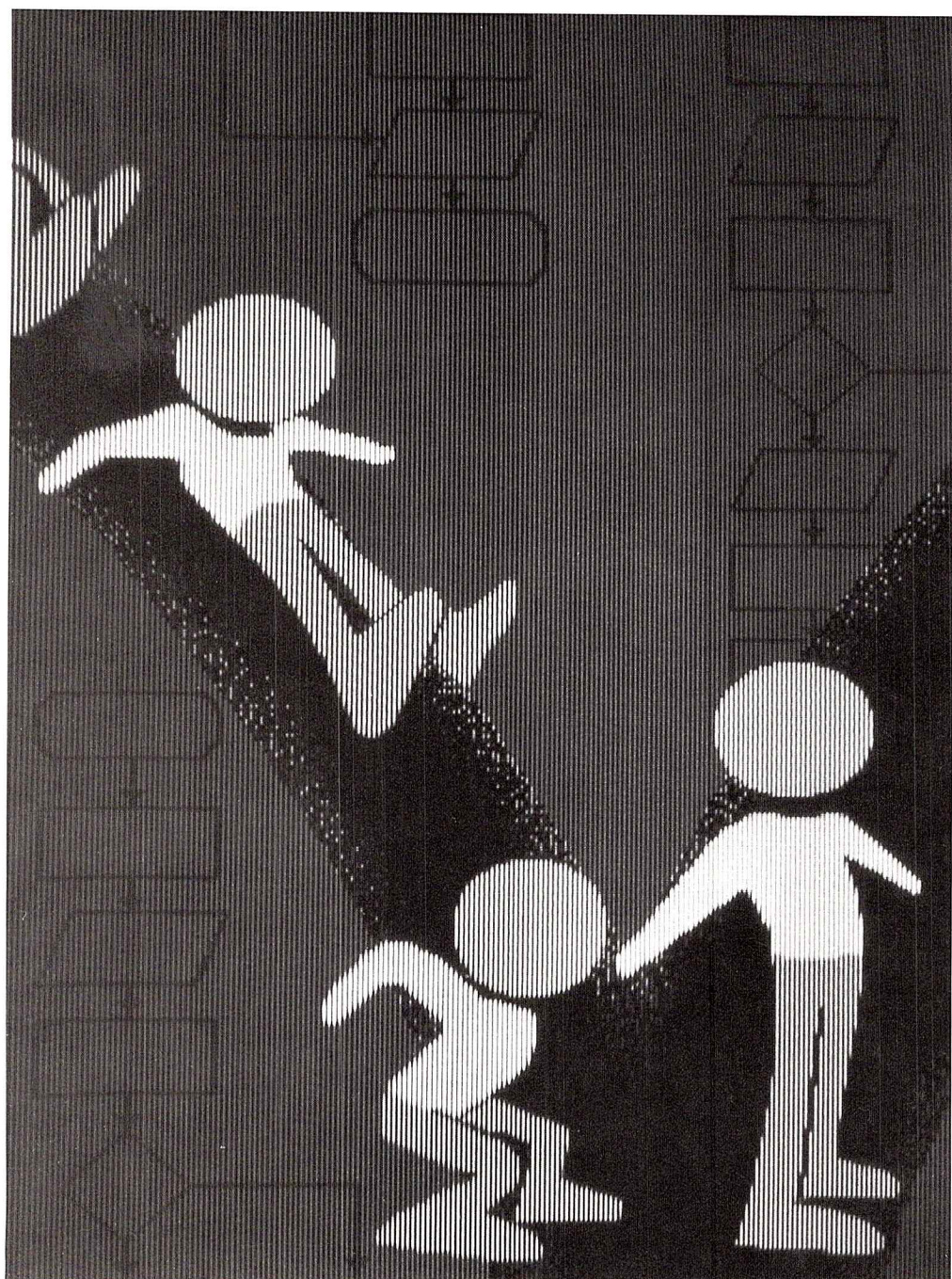
The syntaxes of the FRE function are:

```
FRE(n)  
FRE(" ")
```

In the FRE(*n*) syntax, there are three different functions.

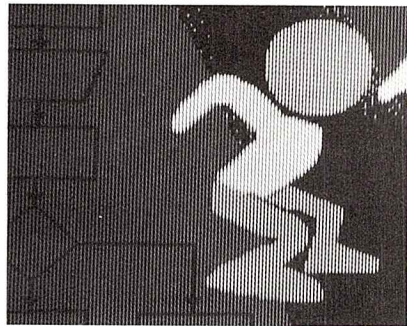
1. If (*n*) is -1, the function returns the number of free bytes available in the heap.
2. If (*n*) is -2, the function returns the number of bytes **never** used by the stack. This does not return the number of free bytes available in the stack. It is used in testing programs to fine-tune the *stack-size* parameter of the CLEAR statement.
3. If (*n*) is any number other than -1 or -2, or if you use the FRE(" ") function, BASIC returns the number of free bytes available in BASIC's data segment.

All versions of the FRE function compact string space.



Chapter 7

Creating Animated Images



This chapter describes the Object Editor, a utility program supplied with Amiga Basic that creates images for manipulation by Amiga Basic animation routines. It includes both an overview of the Object Editor and step-by-step instructions for creating an image.

Overview

Amiga Basic implements the animation facilities built into the Amiga system through program statements and the Object Editor. The COLLISION and OBJECT statements (described in Chapter 8) manipulate images in the output window. The Object Editor defines these images (or *objects*, as they are referred to throughout this book).

With the Object Editor, you can:

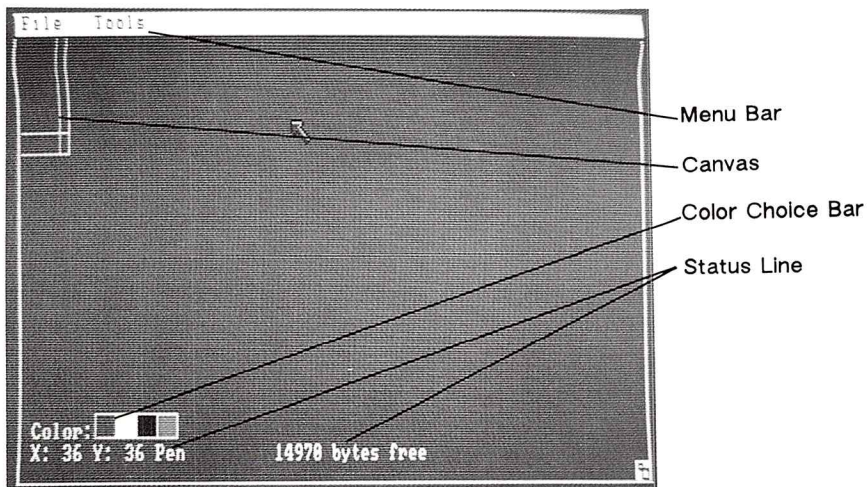
- instantly create ovals, rectangles, and lines by moving the mouse between two points on the Object Editor *canvas*, which is the portion of the Output window where you create the object.
- draw free-form across the canvas with the Object Editor pen
- select colors that form the borders of the object you create
- paint the interior of the objects with the color you desire
- erase and edit the images as required

After creating an object, you save it in a file whose name you specify; the file contains the static attributes (including the size, shape, and color) of the object. To animate the object from a program, open the file, read the contents as a string, and then use the OBJECT.SHAPE to define the object to your program. For an example of statements that do this, see the OBJECT.SHAPE description in Chapter 8 of this manual.

Note: The Object Editor assigns attributes to objects to ensure that, during program execution, they collide both with each other and with the border of the window. You can change this initial setting using an OBJECT.HIT statement (described in Chapter 8) in your program.

The Editor Window

This section explains the layout of the Object Editor window (shown below), where you create draw your objects.



The following paragraphs explain the items in the window:

Menu Bar

Two menus are available: File and Tools. The Tools menu provides several methods of creating images. The File menu provides a means of retrieving and saving the object files you create; these menus are described in the next section.

Canvas

The Canvas, located in the upper lefthand corner, is where you create and color (as well as erase) objects.

You can increase the size of the canvas by placing the pointer in the Sizing Gadget and, while holding down the mouse Selection button, move the mouse until the canvas reaches the desired size. If the new size uses more memory than is available, the following message appears:

OUT OF MEMORY: decrease picture size

When this happens, reduce the size of the canvas before continuing. If you are creating a sprite (a sprite is one of two types of objects you can create, and is described later in this chapter), you cannot increase the width beyond the size displayed (16 pixels, from 0 to 15); you can, however, increase the height.

Color Choice Bar

The Color Choice Bar provides the means of changing the paint and border colors for objects. To change the color, move the pointer over the desired color and click the Selection button. The characters in the word *Color* that appear next to the bar change to the color you select.

The number of color choices in the Choice Bar depend on the depth of the screen, as determined by the *depth* parameter in the SCREEN statement (see Chapter 8 for a description of this statement).

Status Line

To the left are the X and Y coordinates; they indicate the position in the canvas where the Selection button was last pressed. Next, the current Tools selection item (Pen, Oval, Line, Rectangle, Paint, or Eraser) appears. To the right of this item is the number of bytes available to expand the canvas, or, during an error condition, the number of bytes you must reduce the canvas before continuing.

The Editor Menus

The following table summarizes the items in the File menu.

Item	Function
New	Erases the screen and restores the canvas to its original dimensions if they have been changed.
Open	Prompts you for the name of an existing file. You specify the name of any file previously created through the Object Editor and press RETURN.
Save	Saves the file under the same name as it was opened. The Object Editor prompts you for a file name if you previously chose New. Enter the name and press RETURN.
Save as	Prompts for a file name. Specify a name and press RETURN.
Quit	Causes an exit from the Object Editor and returns you to Amiga Basic.

The following table summarizes the items in the Tools menus.

Item	Function
Pen	Allows free-form drawing.
Line	Draws a straight line between two points.
Oval	Draws an egg-shaped image.
Rectangle	Draws a rectangle.
Erase	Removes images from the canvas.
Paint	Permits coloring the interior of an image with the current color choice

A Note about Bobs and Sprites

The Amiga system recognizes two types of objects; Amiga terminology refers to these objects as *sprites* and *bobs*. The Object Editor prompts you to select either a sprite or a bob before you can define the object. Therefore, you must be aware of the differences between these two object types before defining one. (If you are already familiar with these differences, skip to the next section of the chapter.)

The following table summarizes the major difference between sprites and bobs:

Bobs

Move slower than sprites.

Size is limited only by memory available.

Full set of colors allowed.

All bobs can be displayed.

Any screen depth is allowed

Sprites

Move faster than bobs.

Width must be 16.

Only 3 colors allowed.

Only four sprites with different colors can be shown on the same line at the same time.

Screen depth must be 2. The depth corresponds to the value specified for the *depth* parameter of the SCREEN statement; see SCREEN in Chapter 8 for details.

For details on bobs and sprites, see the Graphics Animation Routines chapter in the *Amiga ROM Kernel Manual*.

How to Create Objects

The Object Editor resides on the Amiga Basic disk under the name *objedit.bas*. You open the editor and start operations just as you would any other Amiga Basic program (Chapter 2 gives the steps to achieve this.) Then, follow the steps listed below:

1. Once you've opened the Object Editor, the following prompt appears:

Enter 1 if you want to edit sprites
Enter 0 if you want to edit bobs >

Make the desired selection and press RETURN.

2. Next, the Object Editor window appears. From the Files menu, select New (to create a new object) or Open (to modify an existing object).
3. From the Tools menu, choose how you want to create the image: drawing free-form with the pointer, or by drawing an oval, rectangle, or line. Choose Erase to remove any part of the object.

Move the pointer to the starting position on the canvas, press the Selection button and hold it down, move the pointer to the end position, and then release the button. The drawing or erasure stops when the pointer moves outside the frame and resumes when it returns.

Note that when creating an oval, a rectangle appears on the canvas; upon releasing the button, an oval replaces this rectangle.

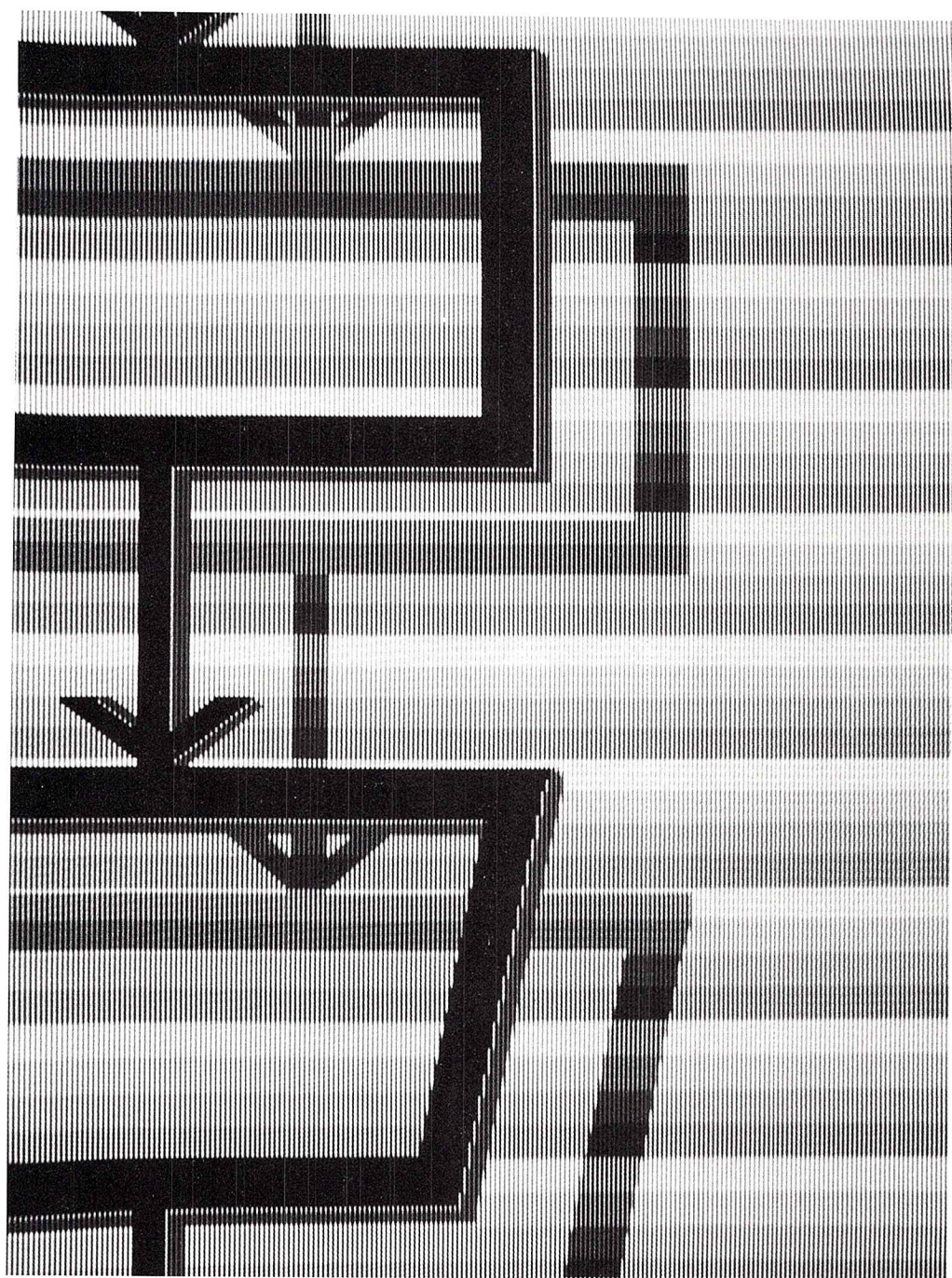
4. To change colors, move the pointer to the color choice bar at the bottom of the screen, and then click the Selection button. The Object Editor then outlines each new image created on the screen with this color.
5. To paint the interior of an image, choose the desired color from the choice bar; then choose Paint from the Tools menu, move the pointer to the region you want to paint, and press the mouse button.

The area you paint should be entirely surrounded by an outline. Otherwise, if a broken border exists, the color "leaks" out into the surrounding area.

6. To make the canvas bigger, place the pointer in the Sizing Gadget, hold down the Selection button, and move the mouse until the canvas reaches the desired size.

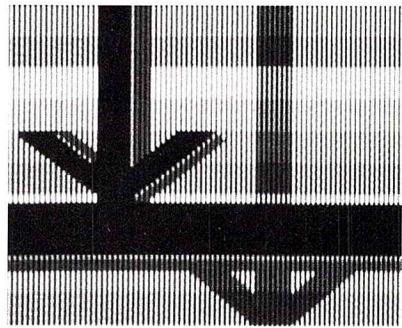
Amiga Basic treats the canvas as one object, regardless of the number of distinct images drawn on it. Multiple objects must be drawn on separate canvases and saved in distinct files.

7. After completing the object, choose Save As (when creating a new object) or Save (when editing an existing object).



Chapter 8

Amiga Basic Reference



The first part of this chapter describes the elements of the Amiga Basic language and the syntax and grammar that applies to the language. The second part is the Statement and Function Directory.

Character Set

The Amiga Basic character set is composed of alphabetic, numeric, and special characters. These are the only characters that Amiga Basic recognizes. There are many other characters that can be displayed or printed, but they have no special meaning to Amiga Basic.

The Amiga Basic alphabetic characters include all the uppercase and lowercase letters of the American English alphabet. Numeric characters are the digits 0 through 9. The following list shows the special characters that are recognized by Amiga Basic.

Character	Name or Function
	Blank
=	Equal sign or assignment symbol
+	Plus sign
-	Minus sign
*	Asterisk or multiplication symbol
/	Slash or division symbol
^	Up arrow or exponential symbol
(Left parenthesis
)	Right parenthesis
%	Percent sign
#	Number (or pound) sign
\$	Dollar sign
!	Exclamation point
[Left bracket
]	Right bracket
,	Comma
.	Period or decimal point
'	Single quotation mark (apostrophe)
;	Semicolon
:	Colon
&	Ampersand
?	Question mark

Character	Name or Function
<	Less than
>	Greater than
\	Backslash or integer division symbol
@	At-sign
_	Underscore
<RETURN>	Terminates input of a line
"	Double quotation mark

The following list shows the Amiga-key characters that are used in Amiga Basic.

Key Combination	Function
Amiga-period(.)	Interrupts program execution and returns to Amiga Basic command level
Amiga-S	Suspends program execution.
Amiga-T	Executes the next statement of the program.
Amiga-C	Executes the "Copy" edit function.
Amiga-P	Executes the "Paste" edit function.
Amiga-X	Executes the "Cut" edit function.
Amiga-R	Executes the "Start" run function.
Amiga-L	Executes the "Show" List window function.

The Amiga Basic Line

Amiga Basic program lines have the following format:

```
[nnnnn] statement [:statement...] [comment] <RETURN>
```

or

```
[alpha-num-label:] statement 1 [:statement 2...] [comment] <RETURN>
```


The *nnnnn* (which specifies the line number) must be an integer between 0 and 65529.

The *alpha-num-label* is any combination of letters, digits, and periods that starts with a letter and is followed (with no intervening spaces) by a colon (:).

A *comment* is a non-executing statement or characters that you may put in your programs to help clarify the program's operation and purpose.

As you can see, Amiga Basic program lines can begin with a line number, an alphanumeric label, neither, or both, and must end with a carriage return. A program line can contain a maximum of 255 characters. More than one Amiga Basic statement can be placed on a line, but each must be separated from the last by a colon. Program lines are entered into a program by pressing the Return key. This carriage return is an invisible part of the line format.

Line numbers and labels are pointers used to document the program (make it more easily understood) or to redirect program flow, as with the GOSUB statement.

If, for example, you want a specific part of a program to run only when a certain condition is met, you could write the following program:

```
IF Account$<>" " THEN GOSUB Design
```

The interpreter searches for a line with the label Design: and executes the subroutine beginning with that line. Note that no colon is needed for Design in the GOSUB statement.

Label Definitions

Alphanumeric line labels can contain from 1 to 40 letters, digits, or periods. They must begin with an alphabetical character. This allows the use of mnemonic labels to make your programs easier to read and maintain.

For example, the following line numbers and alphanumeric labels are valid:

Line Numbers	Alphanumeric Labels
100	ALPHA:
65000	A16:
	SCREEN.SUB:

Restrictions

In order to distinguish alphanumeric labels from variables, each alphanumeric label definition must have a colon (:) following it. A legal label cannot have a space between the name and the colon. When you refer to a label in a GOSUB or GOTO or other control statement, do not include the colon as part of the label name. You cannot use any Amiga Basic reserved word as an alphanumeric label.

While the line number 0 is not restricted from use in a program, error-trapping routines use line number 0 to mean that error trapping is to be disabled. Thus,

```
ON ERROR GOTO 0
```

does not branch to line number 0 if an error occurs. Instead, error trapping is disabled by this statement.

Format

A label, a line number, or both a label and a line number can appear on any line. The line number, when present, must always begin in the leftmost column. A label must begin with the first non-blank character following the line number (if present) and end with a colon; a blank cannot exist between the label and the colon.

Alphanumeric labels and line numbers can be intermixed in the same program.

Constants

Constants are the actual values Amiga Basic uses during program execution. There are two types of constants: string and numeric. A string constant is a sequence of alphanumeric characters enclosed in double quotation marks. String constants may be up to 32,767 characters in length.

Numeric constants are positive or negative numbers. There are five types of numeric constants:

Short Integer	Whole numbers between -32768 and +32767. Short integer constants do not contain decimal points.
Long Integer	Whole numbers between -2147483648 and 2147483647. Long integer constants do not contain decimal points.
Fixed-point	Positive or negative real numbers; that is, number constants that contain decimal points.
Floating-point	Positive or negative numbers represented in exponential form (similar to scientific notation). A floating-point constant consists of an optionally signed integer or fixed-point number (the mantissa) followed by the letter E and an optionally signed integer (the exponent). (Double precision floating-point constants are denoted by the letter D instead of E.)
Hex constants	Hexadecimal numbers with the prefix &H.
Octal constants	Octal numbers with the prefix &O or &.

Fixed-point and floating-point constants can be either single precision or double-precision numbers. Single-precision numeric constants are stored with 7 digits of precision (plus the exponent) and printed with up to 7 digits of precision. Double-precision numbers are stored with 16 digits of precision and printed with up to 16 digits of precision. (See Appendix D, Internal Representation of Numbers, for details on the internal format of numbers. A single precision constant is any numeric constant that has one of the following properties:

- Seven or fewer digits
- Exponential form denoted by E
- A trailing exclamation point (!)

A double precision constant is any numeric constant that has one of the following properties:

- Eight or more digits
- Exponential form denoted by D
- A trailing declaration character (#)

The following are examples of numeric constants:

Single Precision	Double Precision
46.8	345692811
-1.09E-6	-1.09432D-06
3489.0	3489.0#
22.5!	7654321.1234

Numeric constants in Amiga Basic cannot contain commas.

Variables

Variables represent values that are used in a program. As with constants, there are two types of variables: numeric and string. A numeric variable can only be assigned a value that is a number. A string variable can only be assigned a character string value. You can assign a value to a variable, or it can be assigned as the result of calculations in the program. Before a variable is assigned a value, its value is zero (numeric variables) or null (string variables).

Variable Names

A variable name can contain as many as 40 characters. The characters allowed in a variable name are letters, numbers, and the decimal point. The first character in a variable name must be a letter. Special type declaration characters are also allowed (see “Declaring Variable Types” in this section).

Variable names are not case-sensitive. That means that variables with the names ALPHA, alpha, and AlPhA are the same variable.

If a variable begins with FN, Amiga Basic assumes it to be a call to a user-defined function. (See “DEF FN” in the Statement and Function Directory that follows for more information on user-defined functions.)

Reserved Words

Reserved words are words that have special meaning in Amiga Basic. They include the names of all Amiga Basic commands, statements, functions, and operators. Examples include GOTO, PRINT, and TAN. Always separate reserved words from data or other elements of an Amiga Basic statement with spaces. Reserved words cannot be used as variable names. Reserved words can be entered in either uppercase or lowercase. A complete list of reserved words is given in Appendix C, “Amiga Basic Reserved Words.”

While a variable name cannot be a reserved word, a reserved word embedded in a variable name is allowed.

Declaring Variable Types

Variable names can be declared either as numeric values or as string values. String variable names can be written with a dollar sign (\$) as the last character. For example:

```
LET A$ = "SALES REPORT"
```

The dollar sign is a variable type declaration character; that is, it "declares" that the variable will represent as String.

You can assign a numeric value certain properties by appending a trailing declaration character to its variable name. You can declare the value to be a short integer or a long integer a with single-precision or double-precision value. Computations with double-precision variables are more accurate than single-precision variables. However, double-precision variables take up more memory space than single-precision precision variables.

The default type for a numeric variable is single precision.

The trailing declaration characters for numeric variables and the memory requirements (in bytes) for storing each variable type are as follows:

%	SHORT Integer	2
&	LONG Integer	4
!	Single precision	4
#	Double precision	8
\$	String	5 bytes plus the contents of the string.

Instead of using the trailing declaration characters, you can include **DEFINT**, **DEFLNG**, **DEFSTR**, **DEFDBL**, and **DEFSNG** statements in a program to relate the starting letter of a variable name to a variable type. Each time you declare a variable name beginning with the specified letter, Amiga Basic assumes the variable type you specified in the *DEFTYPE*

statement. (These statements are described in the DEFINT section later in this chapter.)

Array Variables

An array is a group of values of the same type, referenced by a single variable name. The individual values in an array are called elements. Array elements are variables also. They can be used in any Amiga Basic statement or function that uses variables. Declaring the name and type of an array and setting the number of elements in the array is known as *dimensioning* the array.

Each element in an array is referenced by an array variable that is subscripted with an integer or an integer expression. An array variable name has as many subscripts as there are dimensions in the array. For example, `V(10)` would reference a value in a one-dimension array, `T(1,4)` would reference a value in a two-dimension array, and so on. Note that the array variable `T(n)` and the “simple” variable `T` are not the same variable. The maximum number of dimensions for an array is 255. The maximum number of elements per dimension is 32,768.

Individual elements of string arrays need not be the same length.

Array elements, like numeric variables, require a certain amount of memory space, depending on the variable type. The memory requirements for storing arrays are the same as for variables, each element of the array requiring as much as the same type variable.

Type Conversion

When necessary, Amiga Basic will convert a numeric constant from one type to another. Keep the following rules in mind.

If a numeric constant of one type is assigned to a numeric variable of a different type, the numeric constant is stored as the type declared in the

variable name. (If a string variable is assigned to a numeric value or vice versa, a "Type mismatch" error message is generated.)

During expression evaluation, all of the operands in an arithmetic or relational operation are converted to the same degree of precision; that is, the degree of the most precise operand. Also, the result of an arithmetic operation is returned to this degree of precision.

Logical operators convert their operands to integers and return an integer result. The operand must be in the range applicable to the short integer or long integer specified.

When a floating-point value is converted to an integer, the fractional portion is rounded.

Expressions and Operators

An expression is a combination of constants, variables, and other expressions with operators. Expressions are "evaluated" by the interpreter to produce a string or numeric value. Operators perform mathematical or logical operations on values. The operators provided by Amiga Basic can be divided into four categories:

- Arithmetic
- Relational
- Logical
- Functional

Hierarchy of Operations

The Amiga Basic operators have an order of precedence; that is, when several operations take place within the same program statement, certain operations are executed before others. If the operations are of the same level, the leftmost one is executed first, the rightmost last. The following is the order in which operations are executed:

1. Exponentiation
2. Unary Negation
3. Multiplication and Floating-point Division
4. Integer Division
5. Modulo Arithmetic
6. Addition and Subtraction
7. Relational Operators
8. NOT
9. AND
10. OR and XOR
11. EQV
12. IMP

Arithmetic Operators

The Amiga Basic arithmetic operators are listed in the following table in order of operational precedence:

Operator	Operation	Sample Expression
\wedge	Exponentiation	X^Y
$-$	Unary Negation	$-X$
$*, /$	Multiplication	$X * Y$
	Floating point Division	X / Y
\backslash	Integer Division	$X \backslash Y$
MOD	Modulo Arithmetic	$Y \text{ MOD } Z$
$+, -$	Addition, Subtraction	$X + Y, X - Y$

To change the order in which the operations are performed, use parentheses. Operations within parentheses are performed first. Inside parentheses, the usual order of operation is maintained.

Amiga Basic expressions look somewhat different from their algebraic equivalents. Here are some sample algebraic expressions and their Amiga Basic counterparts:

Algebraic Expression	Amiga Basic Expression
$\frac{X - Z}{Y}$	<code>(X - Z) / Y</code>
$\frac{XY}{Z}$	<code>X * Y / Z</code>
$\frac{X + Y}{Z}$	<code>(X + Y) / Z</code>
$(X^2)^Y$	<code>(X^2)^Y</code>
X^{Y^Z}	<code>X^(Y^Z)</code>
$X(-Y)$	<code>X*(-Y)</code>

Integer Division

Integer division is denoted by the backslash (\) instead of the slash (/); the slash indicates floating-point division. The operands of integer division are rounded to integers (for short integers, in the range -32768 to +32767 and for long integers, from -2147483648 to 2147483647) before the division is performed, and the quotient is truncated to an integer.

For example:

```
X=10/4
Y=25.68\6.99
PRINT X,Y
```

2 3

Modulo Arithmetic

Modulo arithmetic is denoted by the operator MOD. Modulo arithmetic provides the integer remainder of an integer division.

For example:

10.4 MOD 4=2	(10\4=2 with a remainder of 2)
25.68 MOD 6.99=5	(26\7=3 with a remainder of 5)

Note that Amiga Basic rounds both the divisor and the dividend to integers for the MOD operation.

Overflow and Division by Zero

If a division by zero is encountered during the evaluation of an expression, the "Division by zero" error message is also displayed, machine infinity (the highest number Amiga Basic can produce) with the sign of the numerator is supplied as the result of the division, and execution continues. If the evaluation of an exponentiation results in zero being raised to a negative power, the "Division by zero" error message is displayed, positive machine infinity is supplied as the result of the exponentiation, and execution continues. If overflow occurs, the "Overflow" error message is displayed, plus or minus infinity is supplied as a result, and execution continues.

Relational Operators

Relational operators are used to compare two values. The result of the comparison is either "true" (-1) or "false" (0). This result can then be used to make a decision regarding program flow (see the "IF...THEN" statement in the Statement and Function Directory). The following table lists the relational operators:

Operator	Relation Tested	Expression
=	Equality	X=Y
< >	Inequality	X < > Y
<	Less than	X < Y
>	Greater than	X > Y
< =	Greater than or equal to	X < = Y
> =	Less than or equal to	X > = Y

(The equal sign is also used to assign a value to a variable. See “LET” in the Statement and Function Directory.) When arithmetic and relational operators are combined in one expression, the arithmetic operation is always performed first.

Logical Operators

Logical operators perform bit manipulation, Boolean operations, or tests on multiple relations. Like relational operators, logical operators can be used to make decisions regarding program flow.

A logical operator returns a result from the combination of true–false operands. The result (in bits) is either “true” (–1) or “false” (0). The true–false combinations and the results of a logical operation are known as *truth tables*. There are six logical operators in Amiga Basic. They are: NOT (logical complement), AND (conjunction), OR (disjunction), XOR (exclusive or), IMP (implication), and EQV (equivalence). Each operator returns results as indicated in the following table. A “T” indicates a true value and an “F” indicates a false value. Operators are listed in order of operational precedence.

Operation	Value	Value	Result
NOT	X		NOT X
	T		F
	F		T
AND	X	Y	X and Y
	T	T	T
	T	F	F
	F	T	F
	F	F	F
OR	X	Y	X OR Y
	T	T	T
	T	F	T
	F	T	T
	F	F	F

Operation	Value	Value	Result
XOR	X	Y	X XOR Y
	T	T	F
	T	F	T
	F	T	T
	F	F	F
IMP	X	Y	X IMP Y
	T	T	T
	T	F	F
	F	T	T
	F	F	T
EQV	X	Y	X EQV Y
	T	T	T
	T	F	F
	F	T	F
	F	F	T

In an expression, logical operations are performed after arithmetic and relational operations. Logical operators convert their operands to 16-bit, signed, two's complement integers in the range applicable to the long integer or short integer specified.

If both operands are supplied as 0 or -1, logical operators return 0 or -1, respectively. The given operation is performed on these integers in bits; that is, each bit of the result is determined by the corresponding bits in the two operands. Thus, it is possible to use logical operators to test bytes for a particular bit pattern. For instance, the AND operator can be used to "mask" all but one of the bits of a status byte. The OR operator can be used to "merge" two bytes to create a particular binary value. The following examples illustrate how the logical operators work:

63 AND 16 = 16	63 = binary 11111 and 16 = binary 10000, so 63 AND 16 = 16.
15 AND 14 = 14	15 = binary 1111 and 14 = binary 1110, so 15 AND 14 = 14 (binary 1110).
-1 AND 8 = 8	-1 = binary 1111111111111111 and 8 = binary 0000, so -1 AND 8 = 8.
4 OR 2 = 6	4 = binary 100 and 2 = binary 10, so 4 OR 2 = 6 (binary 110).
-1 OR -2 = -1	-1 = binary 1111111111111111 and -2 = binary 1111111111111110, so -1 OR -2 = -1. The binary complement of 16 zeroes is sixteen ones, which is the two's complement representation of -1.
NOT X = -(X+1)	The two's complement of any integer is the bit complement plus one.

Functions and Functional Operators

When a function is used in an expression, it calls a predetermined operation that is to be performed on its operands. Amiga Basic has two types of functions: "intrinsic" functions, such as SQR (square root) or SIN (sine) which reside in the system, and user-defined functions that are written by the programmer.

See the Statement and Function Directory for exact description of individual intrinsic functions and "DEF FN".

Using Operators with Strings

A string expression consists of string constants, string variables, and other string expressions combined by operators. There are three classes of operations with strings: concatenation, relational, and functional.

Concatenation

Combining two strings together is called concatenation. The plus symbol (+) is the concatenation operator. Here is an example of the use of the operator:

```
LET A$ = "File" : LET B$ = "name"  
PRINT A$ + B$  
PRINT "New " + A$ + B$  
END
```

These statements display the following on the screen:

```
Filename  
New Filename
```

This example combines the string variables A\$ and B\$ to produce the value "Filename."

Relational Operators

Strings can also be compared using the same relational operators that are used with numbers:

= < > < > < + > =

Using operators with strings is similar to using them with numbers, except that the operands are strings rather than numeric values. String comparisons are made by taking one character at a time from each string and comparing the ASCII codes. The ASCII code system assigns a number value to each character produced by the computer. (See Appendix A, "ASCII Character Codes.") If all the ASCII codes are the same, the strings are equal. If the ASCII codes differ, the lower code number precedes the higher. If during string comparison the end of one string is reached, the shorter string is said to be smaller if they are equal to that point. Leading and trailing blanks are significant.

Here are some examples of true expressions:


```
"AA" < "BB"  
"FILENAME" = "FILENAME"  
"X&" > "X#"  
"CL " > "CL "  
"kg" > "KG"  
"SMYTH" < "SMYTHE"
```

Thus, string comparisons can be used to test string values or to alphabetize strings. All string constants used in comparison expressions must be enclosed in quotation marks.

Statement and Function Directory

Headings

Amiga Basic is a powerful programming language with over one hundred thirty statements and functions. These are presented in alphabetical order and are described as follows:

Syntax	Shows the correct syntax for the statement or function. There are two kinds of syntaxes: one for statements and one for functions. All functions return a value of a particular type and can be used wherever an expression can be used. Unlike functions, statements can appear alone on an Amiga Basic program line or they can be entered in immediate mode where they are considered commands.
Action	Summarizes what the statement or function does.
Remarks	Describes arguments and options in detail, and explains how to use the statement or function.
See also	Cross-references to related statements and functions. Optional section.

Examples	Gives sample commands, programs, and program segments that illustrate the use of the given statement or function. Optional section.
Note	Points out an important caveat or feature. Optional section.
Warning	Alerts the user to problems or dangers associated with use of the given statement or function. Optional section.

The following syntax notation is used in this section:

CAPS	Items in capital letters must be input as shown.
<i>italics</i>	Items in italics are to be supplied by the user.
[]	Items inside square brackets are optional.
...	Items followed by ellipses may be repeated any number of times.
{ }	Braces indicate that the user has a choice between two or more items. One of these items must be chosen unless the entries are also enclosed in square brackets.
	Vertical bars separate the items enclosed in braces discussed above.

All punctuation including commas, parentheses, semicolons, hyphens, and equal signs must be included where shown.

ABS

ABS (X)

Returns the absolute value of the expression X.

Example:

The following example shows the results ABS returns for a positive and a negative number.

```
LET X = 987 : LET Y = -987
PRINT ABS (X), ABS(Y)
```

The results displayed on the screen are as follows:

```
987  987
```

AREA

AREA [STEP](X,Y)

Defines a point of a polygon to be drawn with the AREAFILL statement.

The parameters *x* and *y* specify one of many points that Amiga Basic is to connect in forming a polygon with an AREAFILL statement. The AREAFILL statement ignores all AREA statements in excess of 20.

If STEP is included, *x* and *y* are offsets from the current graphics pen position. Otherwise, they are absolute values specifying a location in the current window.

See also: AREAFILL

AREAFILL

AREAFILL [mode]

Alters the interior of a polygon defined by two or more preceding AREA statements.

The *mode* parameter determines the format of the polygon as shown in the following table.

- | | |
|---|--|
| 0 | Fills the area with the area pattern established by the PATTERN statement. This is the default mode. |
| 1 | Inverts the area. |

Example:

The following statements draw a triangle and fill its interior:

```
AREA (10,10)
AREA STEP (0,40)
AREA STEP (40,-40)
AREAFILL
```

See also: AREA, PATTERN, and COLOR

ASC

ASC(X\$)

Returns a numerical value that is the ASCII code for the first character of the string X\$.

The Amiga Basic character set includes the entire ASCII set, but also contains additional characters. These non-ASCII characters, as well as the standard ASCII characters, may be tested with the ASC function (see Appendix A, "ASCII Character Codes").

See also: CHR\$

Example:

The following demonstrates the use of the ASC function:

```
LET OBJECT$ = "T"
PRINT ASC(OBJECT$)
END
```

This statement prints out the following value:

84

ATN

ATN(X)

Returns the arc tangent of X, where X is in radians. The result is in the range $-\pi/2$ to $\pi/2$ radians.

The evaluation of this function is performed in single precision when the argument is in single precision and in double precision when the argument is in double precision.

Examples:

In the following example, ATN is used in a program that converts numbers to their respective arc tangents.

```
`Arc tangent request program
newnumber:
INPUT "Enter a number ", NUMBER
PRINT "Arc tangent of " NUMBER " is " ATN(NUMBER)
INPUT "If you have another number, enter y ", YORN$
IF YORN$ = "y" GOTO newnumber
END
```

The following example shows the results produced by this program:

```
Enter a number 33
Arc tangent of 33 is 1.540503
If you have another number, enter y y
Enter a number 2
Arc tangent of 2 is 1.107149
If you have another number, enter y n
```


BEEP

BEEP

Sounds the speaker and flashes the display.

The BEEP statement causes a momentary sound. The statement is useful for alerting the user.

Example:

```
IF MemLeft& < 100 THEN
  BEEP
  LOCATE 17,1
  PRINT "OUT OF MEMORY: decrease picture size";
END IF
```

BREAK ON BREAK OFF BREAK STOP

BREAK ON
BREAK OFF
BREAK STOP

Enables, disables, or suspends event trapping based on the user trying to stop program execution.

The BREAK ON statement enables event trapping of user attempts to halt the program (by pressing Amiga-period or selecting the Stop option on the Run menu).

The BREAK OFF statement disables ON BREAK event trapping. Event trapping stops until a subsequent BREAK ON statement is executed. The BREAK STOP statement suspends BREAK event trapping. Event trapping continues, but Amiga Basic does not execute the ON BREAK...GOSUB statement for an event until a subsequent BREAK ON statement is executed.

See also: ON BREAK

Example:

This program fragment illustrates the use of ON BREAK.

```
BREAK ON
ON BREAK GOSUB DIRECTUSER
DIM PAYTIME(99),HRS(99),GROSS(99),FIT(99),FICA(99),STATE(99),NET(99)
LET TOTALEMPLOYEES = 99
OPEN "O",#1,"EmployeePay"
  FOR I=1 TO TOTALEMPLOYEES

WRITE#1,PAYTIME(I),HRS(I),GROSS(I),FIT(I),FICA(I),STATE(I),NET(I)
  NEXT I
CLOSE #1 :BREAK OFF
INPUT "Do you wish to print the payroll now (Y/N)?", ANSWER$
IF ANSWER$ = "Y" THEN BREAK ON: GOSUB PRINTCHECKS
END
DIRECTUSER:
  CLS:BEEP:PRINT "You can't exit program until file is updated."
  RETURN
```

CALL

CALL name [(*argument-list*)]

name [*argument -list*]

(1) Calls an Amiga Basic subprogram as defined by the SUB statement; (2) calls a machine language routine at a fixed address; or (3) calls a machine language LIBRARY routine.

The CALL keyword optional. If CALL is omitted, the parentheses surrounding *argument-list* are also omitted.

Calling Amiga Basic Subprograms Defined by the SUB Statement

You can call subprograms using the SUB statement. Variables are passed by reference. Expressions are passed by value.

For example,

```
SUB ALPHA (x,y) STATIC
END SUB
CALL ALPHA (a,b)
```

See the SUB statement in this chapter and also in Chapter 6 for more information on calling subprograms.

Calling Machine Language Subprograms

The CALL statement is the only way to transfer program flow to an external subroutine. The name identifies a simple variable that contains an address that is the starting point in memory of the subroutine. The name cannot be an array element.

The argument list contains the arguments that are passed to the subroutine. Parameters are passed by value using the standard C-language calling conventions. All parameters must be short integer or long integer, or Amiga Basic issues a "Type mismatch" message. The address of a single or double precision variable can be passed as follows:

```
CALL Routine(VARPTR(x))
```

The address of a string can be passed as follows:

```
CALL Routine(SADD(x$))
```

In the following example, the variable that holds the address of the routine is a short integer (&). (Use a long integer if the address length is 24 bits; a short integer or a single-precision number can't hold a 24-bit address.)

```
a=0: b=0
DIM Code%(100)
FOR I=0 TO 90
  READ Code%(I)
NEXT I
CodeAdr& = VARPTR(Code%(0))
CALL CodeAdr&(a,b)
```

Calling a Machine Language Subroutine from a LIBRARY

Library routines are machine language routines that are bound to Amiga Basic dynamically at runtime.

Library files are special Amiga resource files.

Parameters are passed by value using standard C-language conventions.

Example:

```
LIBRARY "graphics.library"  
CALL Draw(50,60)
```

In the above example, Amiga Basic creates a variable by the name of Draw. It then stores information about where the machine language routine resides in this variable. For this reason, the variable cannot be an integer.

For example, the following call would generate a "Type mismatch" error

```
DEFINT A-Z  
CALL Draw(50,60)
```

but the following call would be acceptable:

```
DEFINT A-Z  
CALL Draw#(50,60)
```

Note that Amiga Basic ignores the trailing declaration character (#) following the routine name when searching the libraries for the routine. So, in the above example, it would search for "Draw," and not "Draw#."

Warning

Because the word CALL can be omitted, a CALL can be executed with the syntax

name argument-list

Such a CALL statement may resemble an alphanumeric label.

Consider the statement

ALPHA: Let A = 5

It is not visually clear whether the statement is calling a subprogram named ALPHA with no argument list, or the statement LET A = 5 is on a line with the label ALPHA:. In such a case, ALPHA: is assumed to be a line label and not a subprogram call with no arguments.

After a THEN or ELSE keyword, CALL is required to distinguish the identifier from a label.

CDBL

CDBL(X)

Converts X to a double-precision number.

Example:

The following example shows the product of two single-precision numbers displayed in single-precision, and then converted to double precision and displayed.

```
A! = 6666 : B! = 100000!  
PRINT A!*B!, "(result printed in single precision)"  
PRINT CDBL(A!*B!), "(result printed in double precision)"
```

The following is displayed on the screen:

```
6.66E+08      (result printed in single precision)  
66660000     (result printed in double precision)
```

CHAIN CHAIN [MERGE] *filespec* [, *expression*] [, [ALL] [, DELETE *range*]]

Executes another program and passes variables to it from the current program.

The *filespec* is the specification of the program that is called.

The *expression* is a line number, or an expression that evaluates to a legal line number, in the called program. It is the starting point for execution of the called program. If it is omitted, execution begins at the first line. An alphanumeric label cannot be used as a starting point.

The MERGE option allows a subroutine to be brought into the Amiga Basic program as an overlay. That is, the current program and the called program are merged, with the called program being appended to the end of the calling program. The called program must be an ASCII file if it is to be merged.

With the ALL option, every variable, except variables which are local to a subprogram in the current program, is passed to the called program. If the ALL option is omitted, the current program must contain a COMMON statement to list the variables that are passed.

If the ALL option is used and the *expression* is not, a comma must hold the place of the *expression*.

CHAIN leaves files opened.

After an overlay is used, it is usually desirable to delete it so that a new overlay may be brought in. To do this, use the DELETE option.

Note

The CHAIN statement with the MERGE option preserves the current OPTION BASE setting.

If the MERGE option is omitted, CHAIN does not preserve variable types or user-defined functions for use by the chained program. That is, any DEFINT, DEFLNG, DEFSNG, DEFSTR, DEFDBL, or DEF FN statements must be restated in the chained program. Also, CHAIN turns off all event trapping. If event trapping is still desired, each event trap must be turned on again after the chain has executed.

When using the MERGE option, user-defined functions should be placed before the *range* deleted by the CHAIN statement in the program.

Otherwise, the user-defined functions are undefined after the merge is complete.

The `DELETE range` consists of a line number or label, a hyphen, and another line number or label. All the lines between the two specified lines, inclusive, are deleted from the program chained from.

See also: `COMMON`, `MERGE`

Example:

This program illustrates the use of the `CHAIN` and `COMMON` statements.

```
COMMON ACCT,BALANCE!,CHARGES( ), DISCOUNT!, CONTACT$  
CHAIN "Receivables"
```

CHR\$

CHR\$(I)

Returns a string whose one character has the ASCII value given by `I` (see Appendix A, "ASCII Character Codes").

`CHR$` is commonly used to send a special character to the screen or a device. For instance, the ASCII code for the bell character (`CHR$(7)`) can be printed to cause the same effect as the `BEEP` statement, or the form feed character (`CHR$(12)`) can be sent to clear the Output window and return the pointer to the home position.

Example:

In the following example, `CHR$` converts the ASCII codes 65 through 90 to their respective ASCII character representation.

```
CLS  
FOR I = 65 TO 90  
PRINT CHR$(I); SPC(1)  
NEXT I
```

The following is displayed on the screen:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

CINT

CINT(X)

Converts X to an integer by rounding the fractional portion.

If X is not in the range -32768 to 32767, an "Overflow" error message is generated. Related to CINT are the CDBL and CSNG functions which convert numbers to the double precision and single precision data types, respectively.

Example:

The following example displays three non-integer numbers, and then displays each number after conversion with CINT.

```
PRINT CINT(-3.5)
PRINT CINT(-3.2)
FOR I = 1 TO 3
  X = RND*10
  PRINT X, "= random number generated by RND, times 10"
  PRINT CINT(X), "= integer portion of the same number"
NEXT I
```

The following is displayed on the screen:

```
-4
-3
1.213501    = random number generated by RND, times 10
1           = integer portion of the same number
6.518611    = random number generated by RND, times 10
6           = integer portion of the same number
8.686811    = random number generated by RND, times 10
9           = integer portion of the same number
```

See also: CLNG, CDBL, CSNG, FIX, INT

CHDIR

CHDIR *string*

Changes the current directory.

The *string* is an expression that identifies the new directory that becomes the current directory.

Example:

```
CHDIR "df1:"    ' Change to the current directory on Device 1
CHDIR "df0:c"   ' Change to Directory C on Device 0
CHDIR "/"       ' Change to parent directory
```

CIRCLE

CIRCLE [STEP](*x,y*),*radius* [,*color-id* [,*start,end* [,*aspect*]]]

Draws a circle or an ellipse with the specified center and radius.

The *x* parameter is the x coordinate for the center of the circle.

The *y* parameter is the y coordinate for the center of the circle.

The STEP option indicates the x and y coordinates are relative to the current coordinates of the pen. For example, if the most recent point referenced were (10,10), CIRCLE STEP(20,15) would reference a point offset 20 from the pen's current x location and offset 15 from the pen's current y location.

The *radius* is the radius of the circle in pixels. The *color-id* specifies the color to be used; it corresponds to the *color-id* in a PALETTE statement. The default color is the current foreground color as set by the COLOR statement.

The *start* and *end* parameters are the start and end angles in radians. The range is $-2*(\text{Pi})$ through $2*(\text{Pi})$. These angles allow the user to specify where a circle or ellipse begins and ends. If the start or end angle is negative, the circle or ellipse is connected to the center point with a line,

and the angles are treated as if they were positive. The start angle may be less than the end angle.

The *aspect* is the aspect ratio, which is the ratio of the width to the height of one pixel. The aspect ratio used by manufacturers of monitors varies. CIRCLE draws a perfect circle if *aspect* is set to the aspect ratio of the monitor; otherwise, CIRCLE draws an ellipse.

The aspect ratio for the standard Amiga monitor (using high resolution and the 640 by 200 screen) is 2.25:1 or approximately .44 (1/2.25), which is the default for *aspect*. If you specify .44 for *aspect*, or omit a specification, a perfect circle is drawn on the Amiga monitor.

Example:

```
CIRCLE (60,60),55
```

The above example draws a circle with a radius of 55 pixels; the center of the circle is located at x coordinate 60 and y coordinate 60.

```
ASPECT = .1                                'Initialize aspect ratio
WHILE ASPECT<20
  CIRCLE(60,60),55,0,,,ASPECT              'Draw an ellipse
  ASPECT = ASPECT*1.4                      'Change aspect ratio
WEND
```

The above example draws a series of ellipses of varying aspect ratios. The 0 parameter specifies the color; here, the Amiga system background color of blue would apply unless overridden by a PALETTE statement.

CLEAR

CLEAR [,basicData] [,stack]

Sets all numeric variables to zero and all string variables to "" and allocates memory to the Amiga Basic data area and to the system stack. Closes all files and resets all DEF FN, DEFINT, DEFLNG, DEFSNG, DEFDBL, and DEFSTR statements.

basicData is a numeric expression that specifies the amount of memory to be allocated to Amiga Basic program text, variables, string, and file data blocks; the numeric expression must be 1024 bytes or greater. If this parameter is omitted, Amiga Basic allocates the current value.

stack is a numeric expression that specifies the amount of memory to be allocated to the system stack; the numeric expression must be 1024 bytes or greater. If this parameter is omitted, Amiga Basic allocates the current value.

See also: FRE

Examples:

```
CLEAR  
CLEAR, 20000  
CLEAR, 2000  
CLEAR, 20000, 2000
```

CLOSE

CLOSE [[#]*filename*[, [#]*filename* ...]]

Concludes I/O to a file. The CLOSE statement complements the OPEN statement.

The *filename* is the number with which the file was opened. A CLOSE with no arguments closes all open files. The association between a particular file and the *filename* terminates upon execution of a CLOSE statement. The file may then be reopened using the same or a different *filename*; likewise, that *filename* can be reused to open any file.

A CLOSE for a sequential output file writes the final buffer of output. When Amiga Basic performs sequential file I/O, it uses a holding area, called a buffer, to build a worthwhile load before transferring data. If the buffer is not yet full, the CLOSE statement assures that the partial load is transferred.

The END, SYSTEM, CLEAR, and RESET statements and the NEW command always close all disk files automatically. (STOP does not close disk files.)

See also: CLEAR, END, NEW, OPEN, RESET, STOP, SYSTEM

Example:

This is a fragment of a program that opens an existing file, gets data from it, updates it, and returns it.

```
OPEN "Payables" AS #2 LEN = 80
  FIELD #2, 30 AS FIRM$, 30 AS ADDR$, 10 AS OWE$, 10 AS DAY$
  GET #2, ACCOUNT
    LET DEBT! = CVS(OWE$)
    LET DEBT! = DEBT! + CHARGES! - PAID)
    LSET OWE$ = MKS$(DEBT!)
  PUT #2 ACCOUNT
CLOSE #2
PRINT "Account #";ACCOUNT;" updated"
```

CLS Statement

CLS

Erases the contents of the current Output window and sets the pen position to the upper left-hand corner of the Output window.

The CLS statement clears the current Output window only and not other Output windows.

Example:

```
CLS
```

COLLISION

COLLISION(*object-id*)

Amiga Basic maintains a queue of collisions that have occurred and have not yet been reported to the program. Amiga Basic can remember only 16 collisions at one time. After the sixteenth collision, it discards any new collision information. Each call of COLLISION removes one item from this queue of collisions.

The *object-Id* corresponds to the *object-id* in an OBJECT.SHAPE statement; it identifies the object being tested. The number can range from 1 to n. If *object-Id* is 0, the function returns the identification number of an object that collides with another object without removing any information from the collision queue. If *object-Id* is -1, the function returns the identification number of the window in which the collision identified by COLLISIONS(0) occurred.

If *object-Id* is non-zero, the function returns the identification number of an object that collided with *object-id*, and removes the information from the collision queue.

If the function returns a negative number from -1 through -4, the *object-Id* collided with one of the four window borders, as indicated below.

- 1 Top border
- 2 Left border
- 3 Bottom border
- 4 Right border

See also: OBJECT.SHAPE for an example.

COLLISION ON COLLISION OFF COLLISION STOP

COLLISION ON
COLLISION OFF
COLLISION STOP

Enables, disables, or suspends COLLISION event trapping. A COLLISION occurs when an object defined by the OBJECT.SHAPE statement collides with another object or the window border. Use the COLLISION function to determine which object collided.

The COLLISION ON statement enables COLLISION event trapping by the ON COLLISION...GOSUB statement.

The COLLISION OFF statement stops event trapping by the ON COLLISION...GOSUB statement; Amiga Basic does not record any collision until a subsequent COLLISION ON statement is executed. The COLLISION STOP statement suspends COLLISION event trapping. Event trapping continues, but Amiga Basic does not execute the ON COLLISION...GOSUB for an event until a subsequent COLLISION ON statement is executed.

See also: COLLISION, "Event Trapping" in Chapter 6, "Advanced Topics". See OBJECT.SHAPE for an example.

COLOR

COLOR [*foreground-color-id*] [, *background-color-id*]

Indicates foreground and background colors to be used.

Amiga Basic uses the *foreground-color-id* specification to determine the color for drawing points, lines, area fill and text, and the *background-color-id* to determine area surrounding these items.

The *foreground-color-id* and *background-color-id* each correspond to the *color-id* defined in a PALETTE statement or to the default color-ids of the Amiga system (see the PALETTE statement for more information on the default color-ids).

If a COLOR statement is not specified, and a PALETTE statement doesn't override the system color-ids, Amiga Basic uses the system colors; these colors are initially white in the foreground and blue in the background, or the colors as specified by the user with the Preference Tool from the Workbench.

Example:

```
PALETTE 1,RND,RND,RND
PALETTE 2,RND,RND,RND
COLOR 1,2
```

CLNG

CLNG (*numeric expression*)

Converts a numeric expression to long-integer format, rounding off any fractional part.

COMMON

COMMON *variable-list*

Passes variables to a chained program.

The COMMON statement is used in conjunction with the CHAIN statement. COMMON statements may appear anywhere in a program, though it is recommended that they appear at the beginning. This technique decreases the likelihood that program control will branch before the COMMON statements execute, passing the desired values to the chained program.

The same variable cannot appear in more than one COMMON statement. Array variables are specified by appending parentheses (that is "()") to the variable name. If all variables are to be passed, use CHAIN with the ALL option and omit the COMMON statement.

Some versions of Amiga Basic allow the number of dimensions in the array to be included in the COMMON statement. This implementation accepts that syntax, but ignores the numeric expression itself.

Example:

This program illustrates the use of the CHAIN and COMMON statements.

```
COMMON ACCT,BALANCE!, CHARGES(), DISCOUNT!, CONTACT$  
CHAIN "Receivables"
```

CONT

CONT

Continues program execution after an Amiga-period has been typed or a STOP statement has been executed. It can also be used to continue execution after single stepping.

Execution resumes at the point where the break occurred. If the break occurred after a prompt from an INPUT statement, execution continues with the reprinting of the "?" prompt or the prompt string).

CONT is usually used with STOP for debugging. When execution is stopped, intermediate values may be examined and changed using immediate mode statements. Execution may be resumed with CONT or an immediate mode GOTO, which resumes execution at a specified line number. CONT may be used to continue execution after an error has occurred.

CONT is invalid if the program has been edited during the break.

Example:

This example illustrates the use of the CONT and STOP statements.

```
CHECK! =25: DEBIT! = 9.89  
PRINT CHECK!,DEBIT!  
      STOP  
LET BALANCE! = CHECK! - DEBIT!  
PRINT BALANCE!  
END
```

COS

COS(X)

Returns the cosine of X, where X is in radians.

The evaluation of this function is performed in single precision when the argument is in single precision and in double precision when the argument is in double precision.

Example:

The following example returns the cosine of 1, 100, and 100.

```
PRINT "COSINE OF 1 IS " COS(1)
PRINT "COSINE OF 100 IS " COS(100)
PRINT "COSINE OF 1000 IS " COS(1000)
```

The following is displayed on the screen:

```
COSINE OF 1 IS .5403023
COSINE OF 100 IS .8623189
COSINE OF 1000 IS .5623791
```

CSNG

CSNG(X)

Converts X to a single-precision number.

In the following example, the product of two double-precision numbers is displayed in double-precision, then converted to single precision and displayed.

```
A# = 6666 : B# = 100000
PRINT A#*B#, "(result printed in double precision)"
PRINT CSNG(A#*B#), "(result printed in single precision)"
```

The following is displayed on the screen:

666600000 (result printed in double precision)
6.666E+08 (result printed in single precision)

See also: CDBL, CINT

CSRLIN

CSRLIN

Returns the approximate line number (relative to the top border of the current Output window) of the pen.

The value returned is always equal to or greater than 1.

In determining the line number, CSRLIN uses the height and width of the character "0" as determined by the font of the current Output window. This value is always greater than, or equal to, 1.

CSRLIN is the opposite of the LOCATE statement, which positions the pen.

Example:

The following example records the current line and row numbers, moves the cursor to the bottom of the screen, and prints a message; it then restores the cursor to its original position and prints a message.

```
Y = CSRLIN ' GET CURRENT CURSOR LINE NUMBER (VERTICAL POSITION)
X = POS(0) ' GET CURRENT CURSOR COLUMN NUMBER (HORIZONTAL POSITION)
LOCATE 20,1 ' PLACE CURSOR ON LINE 20, COLUMN 1 (BOTTOM OF SCREEN)
PRINT "THIS PRINTS AT LOCATION 20,1 (BOTTOM OF PAGE)"
LOCATE Y,X ' PLACE CURSOR IN ORIGINAL LOCATION
PRINT "THIS PRINTS AT ORIGINAL LOCATION OF CURSOR"
```

See also: POS, LOCATE

CVI
CVL
CVS
CVD

CVI(*2-byte string*)
CVL(*4-byte string*)
CVL(*4-byte string*)
CVD(*8-byte string*)

Converts random file numeric string values to numeric values.

CVI converts a 2-byte string to a short integer. CVL converts a 4-byte string to a long integer. CVL converts a 4-byte string to a long integer. CVS converts a 4-byte string to a single-precision number, and CVD converts an 8-byte string to a double-precision number.

CVI, CVL, CVS, and CVD can be used with FIELD and GET statements to convert numeric values that are read from a random disk file, from strings into numbers. Use the VAL function instead of CVI, CVL, or CVS to return the numerical value of a string.

Example:

```
OPEN FileName$ FOR INPUT AS 1
ColorSet=CVL(INPUT$(4,1))
DataSet=CVL(INPUT$(4,1))
```

See also: MKI\$, MKL\$, MKS\$, MKD\$, VAL

DATA

DATA constant-list

Stores the numeric and string constants that are accessed by the READ statement.

DATA statements are nonexecutable and may be placed anywhere in the program. A DATA statement may contain as many constants as will fit on a line (separated by commas). Any number of DATA statements may be used in a program. READ statements access DATA statements in order (from the top of the program to the bottom). The data contained in a

DATA line may be thought of as one continuous list of items, regardless of how many items are on a line or where the lines are placed in the program.

The *constant-list* parameter may contain numeric constants in any format, that is, fixed-point, floating-point, or integer. (No numeric expressions are allowed in the list.) String constants in DATA statements must be surrounded by double quotation marks only if they contain commas, colons, or significant leading or trailing spaces. Otherwise, quotation marks are not needed.

The variable type (numeric or string) given in the READ statement must agree with the corresponding constant in the DATA statement.

DATA statements may be reread from the beginning by use of the RESTORE statement.

Example:

```
DIM Pattern0%(3)
DIM Pattern1%(3)
DIM Pattern2%(3)
FOR I=0 TO 3
    READ Pattern0%(I)
    READ Pattern1%(I)
    READ Pattern2%(I)
NEXT I
DATA &HAAAA, &H3333, &HFFFF
DATA &H5555, &H3333, &HFFFF
DATA &HAAAA, &H3333, &HFFFF
DATA &H5555, &H3333, &HFFFF
```

See also: READ, RESTORE

DATE\$

DATE\$

Retrieves the current date.

The DATE\$ function returns a ten-character string in the form *mm-dd-yyyy*.

Example:

```
10 PRINT DATE$          'PRINT SYSTEM DATE
```

The following is displayed on the screen:

```
08-10-1985
```

DECLARE FUNCTION Statement

```
DECLARE FUNCTION id [(param-list)] LIBRARY
```

Causes Amiga Basic to search all libraries opened with the LIBRARY statement for the machine language function *id* in any expression within the program.

See LIBRARY statement for details on opened libraries.

The *id* is any valid Amiga Basic identifier and can optionally contain one of the following trailing declaration characters: (% , & , ! , #). The *id* identifies the name of the machine language function and the type of value it returns.

The *param-list* is a list of parameters for the function. This list is ignored by Amiga Basic, but it is useful for documentation purposes.

If the function is found, Amiga Basic passes all parameters (if any) to the function. The trailing declaration character (if any) of the *id* indicates the type returned by the function. If the *id* doesn't have a trailing declaration character, the standard type identifier rules apply. (See DEFINT for standard type rules.) For example, ALPHA# returns a double-precision result, BETA% returns an integer result, and so on.

See the CALL statement for a description of the conventions for passing parameters.

Example:

```
DECLARE FUNCTION ViewPortAddress&() LIBRARY  
LIBRARY "intuition.library"  
VPA& = ViewPortAddress&(WINDOW(7))
```

This sets the variable VPA& to the value returned by the library function ViewPortAddress&.

See also: CSNG, DEFINT, DEFSNG, LIBRARY, CALL

DEF FN *DEF FN name[(parameter-list)]=function-definition*

Defines a user-written function.

The *name* parameter must be a legal variable name with no spaces between it and DEF FN. When specified in a program, *name* invokes the function being defined.

The *parameter-list* contains the variable names in the function definition that are to be replaced when the program invokes the function. Each name must be separated by a comma. These variables contain the values specified in the corresponding argument variables passed from the program function call.

The *function-definition* is an expression, limited to one line, that performs the operation of the function. Variable names that appear in the expression do not affect program variables with the same name.

When a function is invoked, a variable name specified in both the *function-definition* and the *parameter-list* contain the same values. Otherwise, the current value of the *function-definition* variable is used.

The DEF FN statement can define either numeric or string functions. The function always returns the type specified in the calling statement. However, Amiga Basic issues a "Type mismatch" message if the data type specified in the calling statement does not match the data type specified in the DEF FN statement.

The DEF FN statement must be executed before the function it defines is called. Otherwise, Amiga Basic issues an "Undefined user function" message. You cannot specify a DEF FN statement in either immediate mode or within a subprogram.

DEF FN statements apply only to the program in which they are defined. If a program passes control to a new program with a CHAIN statement, a DEF FN statement in the old program does not apply to the new program.

Example:

```
DEF FNPERCENT(A,B) = (A/B)*100
INPUT "ENTER PORTION OF TOTAL AMOUNT ", PORTION
INPUT "ENTER THE TOTAL ", TOTAL
RESULT = FNPERCENT(PORTION,TOTAL)
PRINT "PERCENTAGE IS ";RESULT;"%"
```

The following is an example of input and output when these statements are executed.

```
ENTER PORTION OF TOTAL AMOUNT 276
ENTER THE TOTAL 1000
PERCENTAGE IS 27.6 %
```

DEFINT
DEFLNG
DEFSNG
DEFDBL
DEFSTR

DEFINT letter-range
DEFLNG letter-range
DEFSNG letter-range
DEFDBL letter-range
DEFSTR letter-range

Relates the beginning letter of a variable name to a variable type (short integer, long integer, single precision, double precision, or string.)

Amiga Basic assumes that any variable name beginning with a letter specified in *letter-range* to be one of the variable types shown below.

Statement Variable	Type	Declaration Character
DEFINT	Short integer	%
DEFLNG	Long integer	&
DEFSNG	Single precision (default)	!
DEFDBL	Double precision	#
DEFSTR	String	\$

A variable name with a trailing declaration character (% , & , ! , \$, or #) takes precedence over these statements. (See “Declaring Variable Types” earlier in this chapter for more information on trailing declaration characters.)

DEF *type* declarations apply only to the program in which they are declared; they are reset upon exit from the program.

Example:

```
DEFLNG a-p,w
```

This statement causes any name beginning with any letter from *a* through *p* and the letter *w* to be treated as long integers.

DELETE

DELETE [*line*][*-line*]

Deletes program lines.

The DELETE statement works with both line numbers and alphanumeric labels. If *line* does not exist, an “Illegal function call” error message is generated.

DIM

DIM [SHARED] *variable-list*

Specifies the maximum values for array variable subscripts, and allocates storage accordingly.

Use the DIM statement when the value of an array's subscript(s) must be greater than 10; otherwise Amiga Basic issues a "Subscript out of range" error message. The minimum value for a subscript is always 0, unless otherwise specified with the OPTION BASE statement.

The DIM statement sets all the elements of the specified arrays to an initial value of zero. The maximum number of dimensions allowed in a DIM statement is 255; the number you can actually specify depends on the amount of memory available.

Specify SHARED to make the variables globally accessible to the main program and to all subprograms. The DIM SHARED statement must be specified only in the main program. Using a DIM SHARED statement lets you avoid duplicating the same SHARED statements among several subprograms.

If the array has already been dimensioned or referenced and that variable is later encountered in a DIM statement, Amiga Basic issues a "Redimensioned array" error message. To avoid this error condition, place DIM statements at the top of a program so that they execute before references to the dimensioned variable are made.

Example:

```
DIM SHARED A,B,C(10,2)
DIM CF(19)
FOR I=1 TO 19
  READ CF(I)
  PRINT CF(I)
NEXT I
DATA 0,2,4,5,7,9,11,0,1,-1, 0,0,0,0,0,0, -12,12,0
```

See also: SHARED

END

END

Terminates program execution, closes all files, and returns to previous mode.

END statements may be placed anywhere in the program to terminate execution. An END statement at the end of a program is optional.

EOF

EOF(*filenumber*)

Tests for the end-of-file condition.

Returns -1 (true) if the end of a sequential input file has been reached. Use EOF to test for end-of-file while reading in data with an INPUT statement, to avoid "Input past end" error messages.

When EOF is used with a random access file, it returns true if the last GET statement was unable to read an entire record. It is true because it was an attempt to read beyond the end of the file.

Example:

This program demonstrates a use of the EOF function.

```
OPEN "I",#1,"INFO"
  LINE INPUT #1, LONG$
  PRINT LONG$
CLOSE #1
OPEN "I",#,"INFO"
  WHILE NOT EOF(1)
    PRINT ASC(INPUT$(1,#1));
    LET C = C + 1: IF C = 10 THEN PRINT: LET C = 0
  WEND
CLOSE #1
END
```

ERASE

ERASE *array-variable-list*

Eliminates arrays from memory.

Arrays may be redimensioned after they are erased, or the previously allocated array space in memory may be used for other purposes. If an attempt is made to redimension an array without first erasing it, a error message is generated.

Example:

```
ERASE BobArray
```

ERR

ERR

ERL

ERL

Returns the error number and the line on which the error occurred.

When an error-handling routine is entered by way of an ON ERROR statement, the function ERR returns the error code for the error, and the function ERL returns the line number of the line in which the error was detected.

If the line with the detected error has no line number, ERL will return the number of the first numbered line preceding the line with the error. ERL will not return line labels. The ERR and ERL functions are usually used in IF...THEN statements to direct program flow in an error-handling routine.

With the Amiga Basic Interpreter, if the statement that caused the error was an immediate mode statement, ERL will return 65535.

See Appendix B, "Error Codes and Error Messages," for a list of the Amiga Basic error codes.

Example:

```
ON ERROR GOTO errorfix
.
.
errorfix:
  IF (ERR=55) AND (ERL=90) THEN CLOSE#1:RESUME
```

ERROR

ERROR *integer-expression*

Simulates the occurrence of an Amiga Basic error, or allows error codes to be defined by the user.

ERROR can be used as a statement (part of a program source line) or as a command (in immediate mode).

The value of the *integer-expression* must be greater than 0 and less than 256. If the value of the *integer-expression* equals an error code already in use by Amiga Basic (see Appendix B, "Error Codes and Error Messages"), the ERROR statement causes the error message for the Amiga Basic error to be printed (unless errors are being trapped).

To define your own error code, use a value that is greater than the highest value used by an Amiga Basic error code. Use the highest values possible to avoid conflicting with duplicate codes in future versions of Amiga Basic. You can write an error handling routine to process the error you define.

If an ERROR statement specifies a code for which no error message has been defined, Amiga Basic responds with an "Unprintable error" error message. Execution of an ERROR statement for which there is no error-handling routine causes an error message to be generated and execution to halt.

Example:

This example shows how ERROR is used in direct mode:

```
ERROR 15
String too long
```

EXP

EXP(X)

Returns e (base of natural logarithms) to the power of X ; that is, 2.7182818284590^X .

If X is greater than 88 (for single-precision numbers) or 709 (for double-precision numbers), an "Overflow" error message is displayed, machine infinity with the appropriate sign is supplied as the result, and execution continues. The evaluation of this function is performed in single precision when the argument is in single precision and in double precision when the argument is in double precision.

Example:

The following example returns e to the power of 0, 1, 2, and 3.

```
FOR I = 0 TO 3
PRINT EXP(I)
NEXT I
```

The following is displayed on the screen:

```
1
2.718282
7.389056
20.08554
```

FIELD

FIELD [#]*filenumber*,*fieldwidth* AS *string-variable*...

Allocates space for variables in a random file buffer.

It is good programming practice to have a FIELD statement follow as closely as possible the statement that opens the file it is defining.

The *filenumber* parameter corresponds to the number specified in OPEN when the file was created. The *fieldwidth* is the number of characters to be allocated to the *string-variable*.

The total number of bytes allocated in a FIELD statement must not exceed the record length that was specified when the file was created with OPEN. Otherwise, a "Field overflow" error message is generated. (The default record length is 128 bytes.)

Any number of FIELD statements may be executed for the same file. All FIELD statements that have been executed will remain in effect at the same time.

Note

Do not use a fielded variable name in an INPUT or LET statement. Once a variable name is fielded, it points to the correct place in the random file buffer. If a subsequent INPUT or LET statement with that variable name is executed, the variable's pointer no longer refers to the random record buffer, but to string space.

See also: GET, LSET, OPEN, PUT, RSET

Example:

This is a fragment of a program that opens an existing file, gets data from it, updates it, returns it, and closes it.

```
OPEN "Payables" AS #2
```

FILES

FILES [*string*]

Lists all files in a given directory.

If you omit *string*, the statement lists all files in the current directory. If *string* contains a directory name, all files in that directory are listed. If *string* contains a filename, it is listed if the file exists.

Example:

```
FILES "df1:"  
FILES "c"
```

FIX

FIX(X)

Returns the truncated integer part of X.

FIX(X) is equivalent to $\text{SGN}(X) * \text{INT}(\text{ABS}(X))$. The difference between FIX and INT is that FIX does not round off negative numbers to their next lower number (see the example below).

Example:

The following example shows the operation of FIX and INT on the same negative, non-integer number.

```
30 PRINT FIX(-58.75)  
40 PRINT INT(-58.75)
```

The following is displayed on the screen:

```
-58  
-59
```

See also: CINT, INT

FOR...NEXT

```
FOR variable=x TO y [STEP z]  
NEXT [variable][,variable...]
```

Performs a series of instructions in a loop a given number of times.

The FOR statement uses *x*, *y*, and *z* as numeric expressions, and *variable* as a counter. The expression *x* is the initial value of the counter. The expression *y* is the final value of the counter.

The program lines following the FOR statement are executed until the NEXT statement is encountered. Then the counter *variable* is adjusted by the amount specified by STEP. A check is performed to see if the value of the counter is now greater than the final value of *y*. If it is not greater, Amiga Basic branches back to the statement after the FOR statement and the process is repeated. If it is greater, execution continues with the statement following the NEXT statement. This is called a FOR...NEXT loop.

If STEP is not specified, the increment is assumed to be one (+1). If STEP is negative, the counter is decreased each time through the loop. The loop is executed until the counter is less than the final value.

A FOR statement without a corresponding NEXT statement will generate a "FOR without NEXT" error message. A NEXT statement without a corresponding FOR statement will generate a "NEXT without FOR" error message.

Nested Loops

FOR...NEXT loops may be nested; that is, a FOR...NEXT loop may be placed within the context of another FOR...NEXT loop. When loops are nested, each loop must have a unique variable name as its counter. The NEXT statement for the inside loop must appear before that for the outside loop.

The variable in the NEXT statement may be omitted, in which case the NEXT statement matches the most recent FOR statement. If a NEXT statement is encountered before its corresponding FOR statement, a "NEXT without FOR" error message is generated and execution is terminated.

Example:

In the following example, the FOR statement produces a loop of 11 repetitions, each printing out the current value of I.

```
FOR I = 0 TO 100 STEP 10
PRINT I;
NEXT I
```

The following is displayed on the screen:

```
0 10 20 30 40 50 60 70 80 90 100
```

FRE

FRE (-1)

FRE (-2)

FRE (x)

Return numbers of free bytes in specified areas.

FRE(-1) returns the total number of free bytes in the system. FRE (-2) returns the number of bytes of stack space that has never been used. FRE(x) where x is not -1 or -2 returns the number of free bytes in Amiga Basic's data segment.

Example:

```
DEF FNMemoryLeft& = FRE(0)-INT((BobRight+16)/16)*2*(BobBottom+1)*5-6
```

See also: CLEAR

GET

GET [#][*filename*][,*recordnumber*]

GET (x1,y1)-(x2,y2),*array-name* [(*index*[,*index*...,*index*])]

Reads a record from a random disk file into a random buffer.

Gets an array of bits from the screen.

The two syntaxes shown above correspond to two different uses of the GET statement. These are called a random file GET and a screen GET, respectively.

Random File GET

In the first form of the statement, the *filenumber* is the number under which the file was created with OPEN. If the *recordnumber* is omitted, the next record (after the last GET) is read into the buffer. The largest possible record number is 16,777,215.

After a GET statement has been executed, the data in *recordnumber* may be accessed directly using fielded variables. (See "Random Access Files" in Chapter 5, "Working With Files and Devices," for details on random file operations.) INPUT# and LINE INPUT# also may be executed to read characters from the random file buffer.

EOF(*filenumber*) may be used after a GET statement to check if the GET statement was beyond the end-of-file.

Screen GET

The second form of the GET statement is used for transferring graphic images. GET obtains an array of bits from the screen, and its counterpart, PUT, places an array of bits on the screen.

The arguments to GET include specification of a rectangular area on the display screen with $(x1,y1)-(x2,y2)$. The two points specify the upper left-hand corner of the rectangle and the lower right-hand corner of the rectangle, respectively.

The *array-name* is the name assigned to the place that will hold the image. The array can be any type except string, and the dimension must be large enough to hold the entire image.

The multiple *index* parameters for an array permit multiple objects in a multidimensional graphic array. This allows looping through different views of an object in rapid succession.

Unless the array is of type integer, the contents of the array after a GET is meaningless when interpreted directly (see below).

The required size of the array, in bytes, is:

$$6 + ((y_2 - y_1 + 1) * 2 * \text{INT}((x_2 - x_1 + 16) / 16)) * D$$

where x and y are the lengths of the horizontal and vertical sides of the rectangle. D is the depth of the screen, for which 2 is the default.

The bytes per element of an array are:

2 bytes for integer
4 bytes for single precision
8 bytes for double precision

For example, assume you want to GET (10,20)-(30,40),ARRAY%. The number of bytes required is $6 + (40 - 20 + 1) * 2 * (\text{INT}((30 - 10) + 16) / 16) * 2$ or 174 bytes. Therefore, you would need an integer array with at least 87 elements.

It is possible to examine the x and y dimensions and even the data itself if an integer array is used. The width, height, and depth of the rectangle can be found in elements 0, 1, and 2 of the array, respectively.

The GET and PUT statements are used together to transfer graphic images to and from the screen. The GET statement transfers the screen image bounded by the rectangle described by the specified points into the array. The PUT statement transfers the image stored in the array onto the screen.

Example:

```
GET (0,0)-(127,127),P
```

See also: PUT

GOSUB...RETURN

GOSUB *line*
RETURN [*line*]

Branches to and returns from a subroutine.

The *line* in the GOSUB statement is the line number or label of the first line of a subroutine. Program control branches to the *line* after a GOSUB statement executes. A RETURN within the GOSUB will return control back to the statement just following the GOSUB statement in the program text.

A subroutine may be called any number of times in a program. A subroutine also may be called from within another subroutine. Such nesting of subroutines is limited only by available memory.

RETURN statements in a subroutine cause Amiga Basic to branch back to the statement following the most recent GOSUB statement.

A subroutine may contain more than one RETURN statement, should logic dictate a return at different points in the subroutine.

The *line* option may be included in the RETURN statement to return to a specific line number or label from the subroutine. This type of return should be used with care, however, because any other GOSUB, WHILE, or FOR statements that were active at the time of the GOSUB will remain active, and error messages such as "FOR without NEXT" may be generated.

Subroutines may appear anywhere in the program, but it is recommended that the subroutine be readily distinguishable from the main program. To prevent inadvertent entry into the subroutine, precede it with a STOP, END, or GOTO statement that directs program control around the subroutine.

```

GOSUB InitGraphics
-
-
-
InitGraphics:
    iDraw = 30
    iErase = 0
RETURN

```

GOTO

GOTO *line*

Branches to a specified line.

If the program statement with the number or label *line* is an executable statement, that statement and those following are executed.

If it is a nonexecutable statement, such as a REM or DATA statement, execution proceeds at the first executable statement encountered after *line*.

It is advisable to use control structures (IF...THEN...ELSE, WHILE ...WEND, and ON...GOTO) in lieu of GOTO statements as a way of branching, because a program with many GOTO statements can be difficult to read and debug.

Example:

```

CheckMouse:
    IF MOUSE(0)=0 THEN CheckMouse
    IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
    IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture:
    PUT(X,Y),P
    X=MOUSE(1): Y=MOUSE(2)
    PUT(X,Y),P
    GOTO CheckMouse

```

HEX\$

HEX\$(X)

Returns a string that represents the hexadecimal value of the decimal argument.

X is rounded to an integer before HEX\$(X) is evaluated.

Example:

The following example prints the decimal and hexadecimal values of 10 through 16.

```
FOR A = 10 TO 16
PRINT A ; HEX$(A)
NEXT A
```

The following is displayed on the screen:

```
10 A
11 B
12 C
13 D
14 E
15 F
16 10
```

IF...GOTO	<i>IF expression GOTO line[ELSE else-clause]</i>
IF...THEN...ELSE	<i>IF expression THEN then-clause[ELSE else-clause]</i>
IF...THEN...ELSE Block	<i>IF expression THEN statementBlock ELSEIF expression THEN statementBlock ELSE statementBlock END IF</i>

Makes a decision regarding program flow based on the result returned by an expression.

The following rules apply to syntax 1 and 2 IF...GOTO and IF..THEN...ELSE statements:

- If the result of the *expression* is true, the *then-clause* or GOTO statement is executed.
- If the result of the *expression* is false, the *then-clause* or GOTO statement is ignored and the *else-clause*, if present, is executed.
- The *then-clause* and the *else-clause*, can be nested; that is, they can contain multiple Amiga Basic statements and functions. However, for Syntax 1 and Syntax 2, the clauses must not exceed one line.
- THEN may be followed by either an Amiga Basic statement, a function, or a label or line number.
- GOTO is always followed by a label or line number.
- If the statement does not contain the same number of ELSE and THEN clauses, each ELSE is matched with the closest unmatched THEN.
- If an IF...THEN statement is followed by a line number or label in immediate mode, an "Undefined line number" error message is generated, unless a statement with the specified line number or label had previously been entered in program edit mode.

The rules that apply to Syntax 1 and 2 also apply to Syntax 3. However, Syntax 3 differs in the following respects:

- The *statementBlock* can contain nested IF-THEN-ELSE blocks. Amiga Basic does not limit nested statements to only one line; *statementBlock* can contain one or more Amiga Basic statements entered on different lines.
- If an *expression* is true, the corresponding THEN *statementBlock* is executed, and program execution resumes at the first statement following the END IF statement.

- If no expressions are true, either (1) program execution resumes at the first statement following the END IF statement or (2) the ELSE *statementBlock* (if present) is executed and program execution resumes at the first statement following the END IF statement.
- The ELSE-IF block is optional; Amiga Basic doesn't limit the number you can specify.
- The ELSE block is optional.
- If anything other than a remark follows on the same line as THEN, Amiga Basic considers it a single-line IF-THEN-ELSE statement.
- In a line containing a block ELSE, ELSE IF, or END IF statement, only a label can precede the statement; otherwise, Amiga Basic issues an error message.

A block IF statement does not have to be the first statement on the line.

Example:

```

INPUT a,b
IF a = 1 THEN
    IF b = 1 THEN
        PRINT "a and b are 1"
    ELSE
        PRINT "a = 1,b <> 1"
    END IF
ELSEIF a > 0 THEN
    IF b > 0 THEN PRINT "both a and b > 0"
    REM---above line is single-line-IF, not Block-IF
    PRINT "a > 0"
ELSE
    PRINT "a <= 0"
    PRINT "we know nothing about b"
END IF

```


INKEY\$

INKEY\$

Returns either a one-character string containing a character read from the keyboard or a nullstring if no character is pending at the keyboard.

No characters are echoed. All characters are passed through to the program except for Amiga-period, which terminates the program.

Note that if an Output window is not active while the program is running, and the user presses a key, the key is ignored and a BEEP will occur, since keystrokes on the Amiga are only directed to the selected window.

Example:

```
GetAKey:
a$=INKEY$
IF a$<>" " THEN
  a$=UCASE$(a$)
  IF a$="Y" THEN Response=1
  IF a$="N" THEN Response=2
  IF a$="C" THEN Response=3
  IF Response=0 THEN BEEP
END IF
IF Response = 0 THEN GOTO GetAKey
PRINT Response
```

See also: SLEEP

INPUT

INPUT[;][*prompt-string*;]*variable-list*

Allows input from the keyboard during program execution.

When an INPUT statement is encountered, program execution pauses and a question mark is printed to indicate the program is waiting for data. If the *prompt-string* is included, the string is printed before the question mark. The required data is then entered at the keyboard.

A comma may be used instead of a semicolon after the prompt string to suppress the question mark. For example, the statement INPUT "ENTER BIRTHDATE",B\$ will print the prompt with no question mark.

The data that is entered is assigned to the variables given in the *variable-list*. The number of data items supplied must be the same as the number of variables in the list. Data items are separated by commas.

The variable names in the list may be numeric or string variable names (including subscripted variables). The type of each data item that is input must agree with the type specified by the variable name. (Strings input to an INPUT statement need not be surrounded by quotation marks.)

Responding to INPUT with too many or too few items or with the wrong type of value (numeric instead of string, etc.) causes the prompt message "?Redo from start" to be generated. No assignment of input values is made until an acceptable response is given.

Example:

The following example shows the use of INPUT to prompt a user to enter values for a conversion program.

```
^ THIS PROGRAM CONVERTS DECIMAL VALUES TO HEXADECIMAL
ANSWER$="Y"
WHILE (ANSWER$="Y")
  INPUT "ENTER DECIMAL NUMBER ", DECIMAL
  PRINT "HEX VALUE OF " DECIMAL "IS " HEX$(DECIMAL)
  PRINT "OCTAL VALUE OF " DECIMAL "IS " OCT$(DECIMAL)
  INPUT "DO YOU WANT TO CONVERT ANOTHER NUMBER? ", ANSWER$
  ANSWER$ = UCASE$(ANSWER$)
WEND
END
```

The following shows an example of some of the results displayed when a user interacts with this program.

```
ENTER DECIMAL NUMBER 16
HEX VALUE OF 16 IS 10
OCTAL VALUE OF 16 IS 20
DO YOU WANT TO CONVERT ANOTHER NUMBER? Y
ENTER DECIMAL NUMBER 31
HEX VALUE OF 31 IS 1F
OCTAL VALUE OF 31 IS 37
DO YOU WANT TO CONVERT ANOTHER NUMBER? N
```

INPUT\$

INPUT\$(X[,[#]*filename*])

Returns a string of X characters, and reads from *filename*. If the *filename* is not specified, the characters are read from the keyboard.

If the keyboard is used for input, no characters are echoed on the screen. All control characters are passed through except Ctrl-C, which is used to interrupt the execution of the INPUT\$ function.

```
objAttributes$ = INPUT$(LOF(1),1)
OBJECT.SHAPE 1,objAttributes$
```

INPUT#

INPUT#*filename,variable-list*

Reads items from a sequential file and assigns them to program variables.

The *filename* corresponds to the number specified when the file was created with OPEN. The *variable-list* contains the variable names to be assigned to the items in the file; the data type specified for the variable names must match the data type of the corresponding items in the file.

The data items in the file should appear just as they would if data were being typed in response to an INPUT statement. Amiga Basic ignores leading spaces, carriage returns, and linefeeds; it processes any other character as the first digit of a number. For numeric items, the next space,

carriage return, linefeed, or comma delimits the last digit of the number from the next item.

For string items, if the first character of a string is a quotation mark ("), a second quotation mark delimits the end of the string (such a string cannot contain an embedded quotation mark). If a quotation mark is not the first character, then a comma, carriage return, linefeed, or the 255th character of the string delimits the end of the string item.

INSTR

INSTR([I,]X\$,Y\$)

Searches for the first occurrence of string Y\$ in X\$, and returns the position at which the match is found. Optional offset I sets the position for starting the search.

If I is greater than the number of characters in X\$ (LEN(X\$)), or if X\$ is null or Y\$ cannot be found, INSTR returns 0. If Y\$ is null, INSTR returns I or 1. X\$ and Y\$ may be string variables, string expressions, or string literals.

Example:

The following statements locate a specific field within a string and then replace it with a new string; INSTR determines the byte location of the field.

```
THIS ROUTINE CHANGES THE ADDRESS FIELD IN RECORD$
RECORD$ ="n:JOHN JONES adr:3633 6TH ST WACO, TX      "
PRINT "RECORD$ =          " RECORD$
OFFSET = INSTR(RECORD$,"adr:") 'FIND START OF ADDRESS adr:
MID$(RECORD$,OFFSET,40) = "adr:222 ELM ST. WAXAHACHIE, TX      "
PRINT "MODIFIED RECORD$ = " RECORD$
```

The following is displayed on the screen:

```
RECORD$ =          n:JOHN JONES adr:3633 6TH ST WACO, TX
MODIFIED RECORD$ =  n:JOHN JONES adr:222 ELM ST. WAXAHACHIE, TX
```

INT

INT(X)

Returns the largest integer less than or equal to X.

Example:

```
PRINT INT(3.4)
X = INT(37.98)
PRINT INT(X)
Y = INT(-32.3)
PRINT INT(Y)
```

The following integers would be printed:

```
3
37
-32
```

See also: CINT, FIX

KILL

KILL *filespec*

Deletes a file from disk.

If a KILL command is given for a file that is currently OPEN, a "File already open" error message is generated. The *filespec* argument is any legal Amiga filename.

Example:

This deletes the file named MailLabels:

```
KILL "MailLabels"
```


LBOUND **UBOUND**

LBOUND(*array-name*[,*dimension*])
UBOUND(*array-name*[,*dimension*])

Returns the lower or upper bounds of the dimensions of an array.

The *array-name* is the name of the array variable to be tested.

The *dimension* parameter is an optional number used when the array is multi-dimensional, and specifies the dimensions of the array being bounded. The optional *dimension* parameter specifies for which dimension to find the bound. The default value is 1.

The lower bounds are the smallest indices for the specified dimension of the array. **LBOUND** returns 0 or 1 depending on whether the **OPTION BASE** is 0 or 1.

Example:

LBOUND and **UBOUND** are particularly useful for determining the size of an array passed to a subprogram. For example, a subprogram could be changed to use these functions rather than explicitly passing upper bounds to the routine:

```
CALL INCREMENT (ARRAY1(0), ARRAY2(), TOTAL())  
.  
.  
SUB INCREMENT (A(2), B(2), C(2)) STATIC  
  FOR I = LBOUND(A,1) TO UBOUND (A,1)  
    FOR J = LBOUND(A,2) TO UBOUND(A,2)  
      C(I,J) = A(I,J) + B(I,J)  
    NEXT J  
  NEXT I  
END SUB
```

LEFT\$

LEFT\$(X\$,I)

Returns a string containing the leftmost I characters of X\$.

I must be in the range 0 to 32767. If I is greater than the number of characters in X\$ (LEN(X\$)), the entire string (X\$) is returned. If I = 0, a null string of length zero is returned. See also: MID\$, RIGHT\$

LEN

LEN(X\$)

Returns the number of characters in X\$. Nonprinting characters and blanks are counted.

Example:

The following routines shows the use of LEN in determining the offset of a field within a string.

```
'THIS ROUTINE EXTRACTS THE ADDRESS a: FROM STRING RECORD$
/
RECORD$ = "n:JOHN JONES ss:5349 12 99 a:3633 6TH ST WACO,TX"
LENGTH = LEN(RECORD$) 'DETERMINE LENGTH OF RECORD
OFFSET = INSTR(RECORD$,"a:") 'FIND START OF ADDRESS a:
RIGHTCHAR = LENGTH - OFFSET - 1
ADDRESS$ = RIGHT$(RECORD$,RIGHTCHAR) 'EXTRACT ADDRESS FROM RECORD$
PRINT ADDRESS$
```

The following is displayed on the screen:

```
3633 6TH ST WACO,TX
```

LET

[LET] *variable=expression*

Assigns the value of an expression to a variable.

Notice that the word LET is optional. The equal sign by itself is sufficient for assigning an expression to a variable name.

Example:

The following example shows the optional nature of LET in variable assignments; lines 10 and 20 perform the same function, even though LET is not specified in line 20.

```
10 LET A = 1 : LET B = 2 : LET C = 3
20 D = 1      :      E = 2 :      F = 3
30 PRINT A B C D E F
```

The following is displayed on the screen:

```
1 2 3 1 2 3
```

LIBRARY

LIBRARY *filename*

LIBRARY CLOSE

LIBRARY opens a library of machine language subprograms and functions to Amiga Basic. LIBRARY CLOSE closes all libraries that have been opened by the LIBRARY statement.

The *filename* is a string expression designating the file where Amiga Basic is to look for machine language functions and subprograms. The LIBRARY statement lets you attach up to five library files to Amiga Basic at a time. Amiga Basic continues to look for subprograms in these libraries until a NEW, RUN, or LIBRARY CLOSE statement is executed. See Appendix F for more information on these statements.

The LIBRARY statement can generate the, "File not found" and the "Out of memory" error messages.

To use the LIBRARY statement, you must create a .bmap file on disk; the file describes the routines in the specified library. See Appendix F for a description of how to create this file.

Example:

```
LIBRARY "graphics.library"
CALL SetDrMd& (WINDOW(8),3)
```

LINE

LINE [[STEP](x1,y1)] - [STEP] (x2,y2),[color-id][,b[f]]

Draws a line or box in the current Output window.

The coordinate for the starting point of the line is (x1,y1); the coordinate for the end point of the line is (x2,y2).

The *color-id* specifies the color to be used; it corresponds to the *color-id* parameter in a PALETTE statement.

With the “,b” option, a box is drawn in the foreground, with the points (x1,y1) and (x2,y2) as opposite corners.

The “,bf” option fills the interior of the box. When out-of-range coordinates are given, the coordinate that is out of range is given the closest legal value. Boxes are drawn and filled in the color given by color.

With STEP, relative rather than absolute coordinates can be given. For example, assume that the most recent point referenced was (10,10). The statement LINE STEP (10,5) would specify a point at (20,15), offset 10 from x1; and offset 5 from y1.

If the STEP option is used for the second coordinate in a LINE statement, it is relative to the first coordinate in the statement.

Example:

```
LINE(0,0)-(120,120),,BF
```

The above statement draws a box and fills it in with the foreground color specified by either the COLOR statement or the Amiga system default.

LINE INPUT

LINE INPUT [;][“prompt-string”;]string-variable

Reads an entire line from the keyboard during program execution and places it in a string variable without using delimiters.

The "*prompt-string*" is a literal that Amiga Basic prints to the screen before input is accepted. Amiga Basic prints question marks only when they are part of *prompt-string*. All input from the end of the *prompt-string* to the carriage return is assigned to the *string-variable*.

If LINE INPUT is immediately followed by a semicolon, the carriage return typed by the user to end the line does not echo a carriage return/linefeed sequence on the screen.

To terminate a LINE INPUT statement, press the AMIGA key on the righthand side of the keyboard and a period.

Example:

This example demonstrates the use of LINE INPUT and LINE INPUT#.

```
OPEN "O",#2,"INFO"
  LINE INPUT "Customer Data?";CUSTOMER$
  PRINT #2,CUSTOMER$
CLOSE #2
OPEN "I",#2,"INFO"
  LINE INPUT #2,CLIENT$
PRINT CLIENT$
END
```

When you run this program, the following is displayed on the screen:

```
Customer Data? Clarissa Dalloway $10.17 Penknife
Clarissa Dalloway $10.17 Penknife
```

LINE INPUT#

LINE INPUT# *filenumber*;*string-variable*

Reads an entire line from a sequential file during program execution and places it in a string variable without using delimiters.

The *filenumber* corresponds to the number assigned to the file when it was created with OPEN. The *string-variable* is the variable name to which Amiga Basic assigns the line.

The carriage–return character delimits each line in the file. `LINE INPUT#` reads only the characters preceding the carriage–return character, and then skips this character and the linefeed character before reading the next line.

This statement is useful if each line in a data file is broken into fields, or if an Amiga Basic program saved in ASCII format is being read as data by another program.

See also: `LINE INPUT`, `SAVE`

Example:

See the example for `LINE INPUT`.

LIST

`LIST [line]`
`LIST [line][–[line]], "filename"`

Lists the program currently in memory to a List window, a file, or a device.

The *line* may be a line number or an alphanumeric label. When a `LIST` command is given, the specified lines appear in the List window.

The second syntax allows the following options:

- If only the first *line* is specified, that line and all following lines are listed.
- If only the second *line* is specified, all lines from the beginning of the program through the specified line are listed.
- If both *line* arguments are specified, the entire range is listed.
- If a *filename* is given in a string expression such as `SCRN:` or `LPT1:`, the listed range is printed on the given device.

See also: “List Window Hints” in Chapter 4, “Editing and Debugging Your Programs.”

Example:

This example produces a List Window and lists the program:

LIST

LLIST

LLIST [*line*][-*line*]

Sends a listing of all or part of the program currently in memory to the printer (PRT:).

The options for LLIST are the same as for LIST, except that there is no optional output device parameter; output is always to the printer (PRT:).

See also: LIST

LOAD

LOAD [*filespec*[,R]]

Loads a file from disk into memory.

If the *filespec* is not included, a requester appears to prompt the user for the correct name of the file to load.

The *filespec* must include the filename that was used when the file was saved.

The R option automatically runs the program after it has been loaded.

LOAD closes all open files and deletes all variables and program lines currently residing in memory before it loads the designated program. However, if the R option is used with LOAD, the program is run after it is loaded, and all open data files are kept open. Thus, LOAD with the R option may be used to chain several programs (or segments of the same program). Information may be passed between the programs using their disk data files.

See also: CHAIN, MERGE, SAVE

LOC

LOC(*filenumber*)

For random disk files, LOC returns the record number of the last record read or written.

For sequential disk files, LOC returns a different number, the increment. The increment is the number of bytes written to or read from the sequential file, divided either by the number of bytes in the default record size for sequential files (128 bytes) or the record size specified in the OPEN statement for that file. Mathematically, this can be expressed as shown below.

Number of Bytes Read or Written \ OPEN statement Record Size
= # Returned by LOC(*filenumber*)

For files opened to KYBD: or COM1, LOC returns the value 1 if any characters are ready to be read from the file. Otherwise, it returns 0.

When a file is opened for sequential input, Amiga Basic reads the first record of the file, so LOC returns 1 even before any input from the file occurs. LOC assumes the *filenumber* is the number under which the file was opened.

LOCATE

LOCATE [*line*] [,*column*]

Positions the pen at a specified column and line in the current Output window.

The value of the *column* and *line* parameters must be equal to or greater than 1; the location they specify is relative to the upper-left corner of the current Output window. If you omit these parameters, Amiga Basic uses the current location of the pen.

In determining the column and line position, LOCATE uses the height and width of the character "0" in the font of the current Output window.

Example:

The following example records the current line and row numbers, moves the cursor to the bottom of the screen, and prints a message; it then restores the cursor to its original position and prints a message.

```
Y = CSRLIN ' GET CURRENT CURSOR LINE NUMBER (VERTICAL POSITION)
X = POS(0) ' GET CURRENT CURSOR COLUMN NUMBER (HORIZONTAL POSITION)
LOCATE 20,1 ' PLACE CURSOR ON LINE 24, ROW 1 (BOTTOM OF SCREEN)
PRINT "THIS PRINTS AT LOCATION 20,1 (BOTTOM OF PAGE)"
LOCATE Y,X ' PLACE CURSOR IN ORIGINAL LOCATION
PRINT "THIS PRINTS AT ORIGINAL LOCATION OF CURSOR"
```

LOF

LOF(*filenumber*)

Returns the length of the file in bytes.

Files opened to SCRN:, KYBD:, or LPT1: always return the value 0.

Example:

```
entireFile$ = INPUT$(LOF(1),1)
```

LOG

LOG(X)

Returns the logarithm of X. X must be greater than zero.

The evaluation of this function is performed in single precision when the argument is in single precision and in double precision when the argument is in double precision.

Example:

The following statements generate the five sets of results by means of the LOG function.

```

10 FOR I = 1 TO 2 STEP .2
20 PRINT "LOG OF ";I "= ";LOG(I)
30 NEXT I
40 END

```

The following is displayed on the screen:

```

LOG OF 1 = 0
LOG OF 1.2 = .1823216
LOG OF 1.4 = .3364723
LOG OF 1.6 = .4700037
LOG OF 1.8 = .5877868

```

LPOS

LPOS(X)

Returns the current position of the line printer's print head within the line printer buffer.

X is a dummy argument. LPOS does not necessarily give the physical position of the print head.

Example:

```
IF LPOS(X) > 60 THEN PRINT CHR$(13)
```

LPRINT

LPRINT [*expression-list*]

LPRINT USING

LPRINT USING *string-expression;expression-list*

Prints data on the line printer.

LPRINT and LPRINT USING are the same as PRINT and PRINT USING except that output goes to the line printer instead of to the screen.

Example:

See the examples in PRINT and PRINT USING.

LSET

LSET *string-variable=string-expression*

Moves data from memory to a random file buffer in preparation for a PUT statement.

If the *string-expression* parameter requires fewer bytes than were fielded to the *string-variable*, LSET left-justifies the string in the field, and RSET right-justifies the string. (Spaces are used to pad the extra positions.) If the string is too long for the field, characters are dropped from the right. Numeric values must be converted to strings with MKI\$, MKL\$, MKS\$, or MKD\$ before they are used with LSET or RSET.

Note

LSET and RSET may also be used with a nonfielded string variable to left-justify or right-justify a string in a given field.

MENU

MENU *menu-id, item-id, state* [,*title-string*]

MENU RESET

MENU (0)

MENU (1)

The statements create custom Menu Bar options and items underneath them, or restore the default Menu Bar.

The functions return the number of the last Menu Bar or menu item selection made.

The *menu-id* is the number assigned to the Menu Bar selection. It can be a value from 1 to 10.

The *item-id* is the number assigned to the menu item underneath the Menu Bar. It can be a value from 0 to 20. If *item-id* is between 1 and 20, it specifies an item in the menu. If *item-id* is 0, it specifies the entire menu.

For the *state* argument, use 0 to disable the menu or menu item, 1 to enable it, or 2 to enable the item *and* place a check mark by it. If the *item-id* is 0, the state takes effect for the entire menu.

The *title-string* is a string assigned to be the title of a custom Menu Bar selection or an item underneath one.

Depending on the *state*, the MENU statement enables or disables menu item *item* in MENU *menu-id*. If the *title-string* argument appears, the item name on the Menu Bar is changed to *title-string*.

The MENU RESET statement restores Amiga Basic's default Menu Bar.

The function syntax MENU(0) returns a number which corresponds to the number of the last Menu Bar selection made. MENU(0) is reset to 0 every time it executes, so the Menu Bar can be polled just like INKEY\$.

The function syntax MENU(1) returns a number which corresponds to the number of the last menu item selected.

This set of MENU statements and functions gives you the tools to build custom menus and menu items in the Menu Bar at the top of the screen. If a MENU ON statement is executed, the user's selection of custom menu items can be trapped with the ON MENU GOSUB statement.

You can override the existing Amiga Basic menu items with the MENU statement.

Example:

The following are examples of menu statements.

```
MENU 1,0,1,"Transactions:"  
MENU 1,1,1,"Deposits"  
MENU 1,2,1,"Withdrawals"  
MENU 1,3,1,"Automatic Payment"  
MENU 1,5,1,"Credit Card Purchases"
```

The following are examples of MENU functions.

```
MenuId=MENU(0)  
MenuItem=MENU(1)
```

See also: MENU ON, ON MENU, SLEEP

MENU ON
MENU OFF
MENU STOP

MENU ON
MENU OFF
MENU STOP

Enables, disables, or suspends trapping MENU events; a MENU event occurs when the user selects a menu item defined by the MENU statement. The MENU function can be used to determine which menu item was selected.

The MENU ON statement enables event trapping.

The MENU OFF statement disables ON MENU event trapping. Event trapping stops until a subsequent MENU ON statement is executed. The MENU STOP statement suspends MENU event trapping. Event trapping continues, but Amiga Basic does not execute the ON MENU...GOSUB statement for an event until a subsequent MENU ON statement is executed.

Example:

```
ON MENU GOSUB CheckMenu
ON MOUSE GOSUB CheckMouse
MENU ON
MOUSE ON
```

See also: MENU, ON MENU, "Event Trapping" in Chapter 6, "Advanced Topics."

MERGE

MERGE *filespec*

Appends a specified disk file to the program currently in memory.

The *filespec* must include the filename used when the file was saved. That file must have been saved in ASCII format to be merged. You can put a file in ASCII format by using the A option to the SAVE command. If it was not saved in ASCII format, a "Bad file mode" error message is generated.

Amiga Basic returns to command level after executing a MERGE command.

Example:

```
MERGE "SortRoutine"
```

MID\$

$\text{MID\$}(string-exp1, n [, m]) = string-exp2$
 $\text{MID\$}(X$, $n [, m]$)$

The statement replaces a portion of one string with another string.

The function returns a string of length m characters from X \$, beginning with the n th character.

In the statement syntax, n and m are integer expressions, and $string-exp1$ and $string-exp2$ are string expressions. The characters in $string-exp1$, beginning at position n , are replaced by the characters in $string-exp2$. If n is greater than the number of characters in X \$ (that is, $\text{LEN}(X\$)$), $\text{MID\$}$ returns a null string.

The optional m refers to the number of characters from $string-exp2$ that are used in the replacement. If m is omitted, all of $string-exp2$ is used. The replacement of characters never exceeds the original length of $string-exp1$. In the function syntax, the values n and m must be in the range 1 to 32767. If m is omitted or if there are fewer than m characters to the right of the n th character, all rightmost characters, beginning with the n th character, are returned.

In the function syntax, the values n and m must be in the range 1 to 32767. If m is omitted or if there are fewer than m characters to the right of the n th character, all rightmost characters, beginning with the n th character, are returned. If n is greater than the number of characters in X \$ (that is, $\text{LEN}(X\$)$), $\text{MID\$}$ returns a null string.

Example:

The following statements locate a specific field within a string and then replace it with a new string.

```
'THIS ROUTINE CHANGES THE ADDRESS FIELD IN RECORD$
/
RECORD$ ="n:JOHN JONES adr:3633 6TH ST WACO, TX          "
PRINT "RECORD$ =          " RECORD$
OFFSET = INSTR(RECORD$,"adr:") 'FIND START OF ADDRESS adr:
MID$(RECORD$,OFFSET,40) = "adr:222 ELM ST. WAXAHACHIE, TX          "
PRINT "MODIFIED RECORD$ = " RECORD$
```

The following is displayed on the screen:

```
RECORD$ =          n:JOHN JONES adr:3633 6TH ST WACO, TX
MODIFIED RECORD$ =  n:JOHN JONES adr:222 ELM ST. WAXAHACHIE, TX
```

MKI\$	<i>MKI\$(short-integer-expression)</i>
MKL\$	<i>MKL\$(long-integer-expression)</i>
MKS\$	<i>MKS\$(single-precision-expression)</i>
MKD\$	<i>MKD\$(double-precision-expression)</i>

Puts numeric values into string variables for insertion into random file buffers.

MKI\$ converts a short integer to a 2-byte string.

MKL\$ converts a long integer to a 4-byte string.

MKS\$ converts a single-precision number to a 4-byte string.

MKD\$ converts a double-precision number to a 8-byte string.

You must convert numeric variables to string variables before placing them in a random file. Use MKI\$, MKL\$, MKD\$, and MKS\$ for this purpose. Then move the variable to the random file buffer using either LSET or RSET, and write the buffer to the file using PUT#.

Instead of converting the binary value to its string representation, like the STR\$ function, MK\$ moves the binary value into a string of the proper length. This greatly reduces the amount of storage required for storing numbers in a file.

Example:

```
PRINT #1, MKI$(Flags);
```

The following example illustrates the use of MKI\$, MKS\$, and MKD\$ with random files.

```
OPEN "AccountInfo" AS #2 LEN = 14
  FIELD #2,8 AS ACCT$,4 AS CHECK$,2 AS DEPOSIT$
  GET #2,1
    LET ACCOUNTNO# = 987654332556#
    LET CHECKING! = 123456!
    LET SAVINGS% = 2500
    LSET ACCT$ = MKD$(ACCOUNTNO#)
    LSET CHECK$ = MKS$(CHECKING!)
    LSET DEPOSIT$ = MKI$(SAVINGS%)
  PUT #2,1
CLOSE #2
END
```

See also: CVI, CVS, CVL, CVD, LSET, RSET, Chapter 5, "Working with Files and Devices."

MOUSE

MOUSE(*n*)

The MOUSE function returns information about the left mouse button and the location of the mouse's cursor within the active window. MOUSE does not monitor the right button, which is used to control the menu (see the MENU function for information on monitoring menu selections).

MOUSE performs seven functions; specify any value from 0 through 6 as the *n* parameter to select the desired function. The functions are described in the sections that follow.

MOUSE(0): Mouse Button Position

MOUSE(0) gives the status of the left mouse button. After executing MOUSE(0), Amiga Basic retains the start and end positions of the mouse until a subsequent MOUSE(0) is executed. Therefore, after detecting the movement of the mouse through MOUSE(0), a program should then use MOUSE(3), MOUSE(4), MOUSE(5), and MOUSE(6) to determine the starting and ending positions.

The following table explains the values returned by MOUSE(0).

Value Returned	Explanation
0	The left MOUSE button is not currently down, and it has not gone down since the last MOUSE(0) function call.
1	The left MOUSE button is not currently down, but the operator clicked the left button once since the last call to MOUSE (0). To determine the start and end points of the selection, use MOUSE(3), MOUSE(4), MOUSE(5), and MOUSE(6).
2	The left MOUSE button is not currently down, but the operator clicked the left button twice since the last call to MOUSE (0). To determine the start and end points of the selection, use MOUSE(3), MOUSE(4), MOUSE(5), and MOUSE(6).
-1	The operator is holding down the left mouse button after clicking it once. The return of this value usually signifies that the mouse is moving.
-2	The operator is holding down the left mouse button after clicking it twice. The return of this value usually signifies that the mouse is moving.

MOUSE(1): Current X Coordinate

MOUSE(1) returns the horizontal (X) coordinate of the mouse cursor the last time the MOUSE(0) function was invoked, regardless of whether the left button is down.

MOUSE(2): Current Y Coordinate

MOUSE(2) returns the vertical (Y) coordinate of the mouse cursor the last time the MOUSE(0) function was invoked, regardless of whether the left button was down.

MOUSE(3): Starting X Coordinate

MOUSE(3) returns the horizontal (X) coordinate of the mouse cursor the last time the left button was pressed before MOUSE(0) was called. Use MOUSE(3) in combination with MOUSE(4) to determine the starting point of a mouse movement.

MOUSE(4): Starting Y Coordinate

MOUSE(4) returns the vertical (Y) coordinate of the mouse cursor the last time the left button was pressed before MOUSE(0) was called.

MOUSE(5): Ending X Coordinate

If the left button was down the last time MOUSE(0) was called, MOUSE(5) returns the horizontal (X) coordinate where the mouse cursor was when MOUSE(0) was called. If the left button was up the last time MOUSE(0) was called, MOUSE(5) returns the horizontal (X) coordinate where the mouse cursor was when the left button was released. Use MOUSE(5) to track the mouse as the operator moves it and to determine the coordinate where movement stops.

MOUSE(6): Ending Y Coordinate

MOUSE(6) works the same way as MOUSE(5), except it returns the vertical (Y) coordinate.

Mouse Example

The following routine checks the movement of the mouse. As the mouse moves, the routine moves a graphic image in array *P* to the new X and Y positions.

```
CheckMouse:
  IF MOUSE(0)=0 THEN CheckMouse
  IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
  IF ABS(Y-MOUSE(2)) < 3 THEN CheckMouse
MovePicture:
  PUT(X,Y),P
  X=MOUSE(1): Y=MOUSE(2)
  PUT(X,Y),P
  GOTO CheckMouse
```

MOUSE ON
MOUSE OFF
MOUSE STOP

MOUSE ON
MOUSE OFF
MOUSE STOP

Enables, disables, or suspends event trapping based on the pressing of the mouse button.

The MOUSE ON statement enables event trapping based on a user's pressing the mouse button.

The MOUSE OFF statement disables ON MOUSE event trapping. Event trapping stops until a subsequent MOUSE ON statement is executed. The MOUSE STOP statement suspends MOUSE event trapping. Event trapping continues, but Amiga Basic does not execute the ON MOUSE...GOSUB statement until a subsequent MOUSE ON statement is executed.

See also: MOUSE, ON MOUSE, "Event Trapping" in Chapter 6, "Advanced Topics."

NAME

NAME *old-filename* AS *new-filename*

Changes the name of a disk file.

Both parameters are string expressions. The *old-filename* must exist and the *new-filename* must not exist. Otherwise, an error results.

Example:

In this example, the file that was formerly named Accounts becomes LEDGER.

```
NAME "Accounts" AS "LEDGER"
```

NEW

NEW

Deletes the program currently in memory and clears all variables and the List window.

NEW is entered in immediate mode or selected from the Project menu to clear memory before entering a new program. If there is a program currently in memory, and that program has been changed since it was loaded, a requester will automatically appear to allow saving of that program. If executed from within a program, NEW causes Amiga Basic to return to edit mode.

NEW closes all files and turns off tracing mode. When you execute NEW, the windows retain their sizes and locations.

NEXT

NEXT [*variable*[,*variable*...]]

Allows a series of instructions to be performed in a loop a given number of times.

See "FOR...NEXT" for a discussion of NEXT usage.

OBJECT.AX

OBJECT.AX *object-id*, *value*

OBJECT.AY

OBJECT.AY *object-id*, *value*

Define the acceleration of an object in the x and y directions.

The *object-id* corresponds to the *object-id* in an OBJECT.SHAPE statement; it identifies the object whose acceleration is to be defined.

The *value* specifies the acceleration rate in number of pixels per second per second.

OBJECT.CLIP

OBJECT.CLIP (*x1*,*y1*)-(*x2*,*y2*)

Defines a rectangle and instructs Amiga Basic not to draw objects outside this area.

The *x1* and *x2* parameters define the left and right boundaries of the rectangle on the x axis, and *y1* and *y2* define the top and bottom boundaries on the y axis. The default value of the CLIP rectangle is the border of the current Output window.

OBJECT.CLOSE

OBJECT.CLOSE [object-id [,object-id...]]

The OBJECT.CLOSE statement releases all memory held by one or more objects when the object is no longer needed.

The *object-id* corresponds to the *object-id* in an OBJECT.SHAPE statement; it identifies the one or more objects in the current Output window that OBJECT.CLOSE will release.

If object-id is not specified, all objects in the current Output window are released.

OBJECT.HIT

OBJECT.HIT *object-id*, [*MeMask*] [,*HitMask*]

Determines collision objects for *object-id*.

The *object-id* corresponds to the *object-id* in an OBJECT.SHAPE statement.

By default, all objects collide with each other and the border. This statement can be used to allow some objects to pass through each other without causing a collision.

MeMask is a 16-bit mask that describes *object-id*. *HitMask* is a 16-bit mask that describes the object that *object-id* is to collide with. If the least significant bit of *Hitmask* is set, *object-id* collides with the border. If the *MeMask* of one object, when logically ANDed to the *HitMask* of another object, produces a non-zero result, *object-id* collides with any object described by *HitMask* and a COLLISION event occurs.

For more information on defining *MeMask* and *HitMask*, see the Using *HitMask* and *MeMask* section of the "Graphics Animation Routines," chapter in the *Amiga ROM Kernel Manual* for details.

Example:

```
OBJECT.SHAPE 1,Asteroid$
OBJECT.SHAPE 2,Ship$
OBJECT.SHAPE 3,Missile$
OBJECT.HIT 1,8,7 'collides with border, ship, missile
OBJECT.HIT 2,2,9 'collides with border, asteroid
OBJECT.HIT 3,4,9 'collides with border, asteroid
```

OBJECT.ON
OBJECT.OFF

OBJECT.ON [*object-id* [,*object-id*...]]
OBJECT.OFF [*object-id* [,*object-id*...]]

These two statements make one or more objects visible or invisible.

The *object-id* corresponds to the *object-id* in an OBJECT.SHAPE statement; it identifies an object within the current Output window that OBJECT.ON or OBJECT.OFF will respectively make visible or invisible.

In OBJECT.ON, if *object-id* is not specified, all objects within the current Output window are made visible. If the object was previously started with an OBJECT.START statement, it moves again.

In OBJECT.OFF, if *object-id* is not specified, all objects within the current Output window are made invisible. This statement halts the object if it was started with OBJECT.START, and prevents future collisions.

Example:

See OBJECT.SHAPE for an example of OBJECT.ON.

See also: OBJECT.START and OBJECT.STOP

OBJECT.PLANES

OBJECT.PLANES *object-id*, [*plane-pick*][,*plane-on-off*]

Sets the bob's planePICK and place-on-off masks. For details see the *Amiga ROM Kernel Manual*.

The *object-id* corresponds to the *object-id* in an OBJECT.SHAPE statement; it identifies an object in the current Output window.

The *plane-pick* and *plane-on-off* can be an integer from 0 to 255. It defaults to the value established by the Object Editor.

OBJECT.PRIORITY

OBJECT.PRIORITY *object-id*, *value*

Sets a priority that determines when an object is drawn in relation to other objects with higher or lower priorities. This statement affects only bobs; it has no effect on sprites.

Two objects assigned the same priority are drawn in random order.

The *object-id* corresponds to the *object-id* in an OBJECT.SHAPE statement; it identifies the object to be drawn.

The *value* is a number from -32768 to 32767 indicating the priority; the higher the value specified, the higher the priority. For example, an object with a priority of 8 is displayed "in front of" objects with a priority of 0 through 7.

OBJECT.SHAPE

Statement Syntax 1

OBJECT.SHAPE *object-id*, *definition*

Syntax 1 of the OBJECT.SHAPE statement defines the shape, colors, location, and other attributes of an object that can be moved around the current Output window. This includes blitter-objects (bobs) and VSprites

as discussed in the "Graphic Animation Routines" chapter of the *Amiga ROM Kernel Manual*.

The *object-id* identifies the object and is referred to by other OBJECT statements; *object-id* can range from 1 to n, where n is only limited by memory available.

The *definition* is a string expression that describes the static attributes (including size, shape, and color) of the object. The Object Editor utility program, written in Amiga Basic and supplied with the system, builds this string expression. See Chapter 7 for information on using this program.

Statement Syntax 2

OBJECT.SHAPE *object-id1*, *object-id2*

Syntax 2 of the OBJECT.SHAPE statement copies the shape of *object-id2* to *object-id1*, creating a new object. Both objects share a significant amount of memory; thus memory requirements for multiple objects is reduced when they are created with Syntax 2.

Even though *object-id2* and *object-id1* share memory, you can specify different attributes to each using other OBJECT statements. Amiga Basic initializes the values assigned to OBJECT.X, OBJECT.Y, OBJECT.VX, OBJECT.VY, OBJECT.AX, and OBJECT.AY to 0 for this purpose.

Example:

```
OPEN "ball" FOR INPUT AS 1
OBJECT.SHAPE 1, INPUT$(LOF(1), 1)
```

In the above example, the static attributes of the object (including the size, shape, and color) are in the file *ball* earlier created by the user with the Object Editor program (see Chapter 7).

The following gives an example of an Amiga Basic routine that starts up and handle collisions of the objects defined in *ball*. Refer to the other sections of this chapter for an explanation of the COLLISION statement and the other *OBJECT* statements.

```

WINDOW 4,"Animation",(310,95)-(580,170),15
ON COLLISION GOSUB BounceOff
COLLISION ON
OPEN "ball" FOR INPUT AS 1 'file created by the Object Editor
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
OBJECT.X 1,10
OBJECT.Y 1,50
OBJECT.VX 1,30
OBJECT.VY 1,30
OBJECT.ON
OBJECT.START
WHILE 1
  SLEEP
WEND
BounceOff:
  saveId = WINDOW(1)
  WINDOW 4
  i=COLLISION(0)
  IF i=0 THEN RETURN
  j=COLLISION(i)
  IF j=-1 OR j=-3 THEN
    'object bounced off left or right border
    OBJECT.VY i,-OBJECT.VY(i)
  ELSE
    'object bounced off top or bottom border
    OBJECT.VX i,-OBJECT.VX(i)
  END IF
  OBJECT.START
  WINDOW saveId
RETURN

```

OBJECT.START

OBJECT.START [*object-id* [,*object-id*...]]

OBJECT.STOP

OBJECT.STOP [*object-id* [,*object-id*...]]

The OBJECT.START statement sets one or more objects into motion.

The OBJECT.STOP statement freezes the motion of one or more objects.

The *object-id* corresponds to the *object-id* in an OBJECT.SHAPE statement; it identifies one or more objects in the current Output window that OBJECT.START or OBJECT.STOP respectively sets into motion or freezes.

In OBJECT.START, if *object-id* is not specified, all objects in the current Output window are set in motion.

In OBJECT.STOP, if *object-id* is not specified, all objects in the current Output window are frozen.

When two objects collide, Amiga Basic does an OBJECT.STOP on both objects. When one object collides with the border, Amiga Basic does an OBJECT.STOP on the object.

Example:

See OBJECT.SHAPE for an example of the OBJECT.START statement.

OBJECT.VX OBJECT.VY

Statement Syntax

OBJECT.VX *object-id*, *value*
OBJECT.VY *object-id*, *value*

Function Syntax

OBJECT.VX(*object-id*)
OBJECT.VY(*object-id*)

The statement defines the velocity of an object in the *x* and *y* directions. The function returns the velocity of an object in the *x* and *y* directions.

The *object-id* corresponds to the *object-id* in an OBJECT.SHAPE statement; it identifies the object to which the velocity applies.

The *value* in the statement defines the velocity in number of pixels per second. The function returns the same value.

Example:

```
OBJECT.VX 1,30  
OBJECT.VY 1,30
```

See also: OBJECT.AX, and OBJECT.AY, and OBJECT.SHAPE for an example of the use of this statement with other OBJECT statements.

OBJECT.X OBJECT.Y

Statement Syntax

```
OBJECT.X object-id, value  
OBJECT.Y object-id, value
```

Function Syntax

```
OBJECT.X(object-id)  
OBJECT.Y(object-id)
```

The statements place the object at a specified position in the Output window, which is the starting point for animation. The functions return the current X and Y coordinates of the upper left-hand corner of the object's rectangle.

The *object-id* corresponds to the *object-id* in an OBJECT.SHAPE statement, it identifies the object whose upper left corner is to be defined

The *value* defines the X or Y coordinate; it can be a numeric expression ranging from -32768 to 32767.

You can use the statement to establish an initial starting point for animation, or to relocate the object in the Output window during execution; animation then resumes at the new starting point.

The OBJECT.X and OBJECT.Y functions return respectively the current X and Y coordinates of the upper left corner of the object's rectangle.

Example:

```
OBJECT.X 1,10  
OBJECT.Y 1,50
```

See OBJECT.SHAPE for an example of the use of this statement with other OBJECT statements.

OCT\$

OCT\$(X)

Returns a string that represents the long-integer value of the decimal argument. X is rounded to an integer before OCT\$(X) is evaluated.

Example:

The following example shows the use of OCT\$ in a decimal conversion program.

```
^ THIS PROGRAM CONVERTS DECIMAL VALUES TO HEXADECIMAL  
ANSWER$="Y"  
WHILE (ANSWER$="Y")  
  INPUT "ENTER DECIMAL NUMBER ", DECIMAL  
  PRINT "OCTAL VALUE OF " DECIMAL "IS " OCT$(DECIMAL)  
  INPUT "DO YOU WANT TO CONVERT ANOTHER NUMBER? ", ANSWER$  
WEND  
END
```

The following shows an example of some of the results displayed when a user interacts with this program.

```
ENTER DECIMAL NUMBER 16  
OCTAL VALUE OF 16 IS 20
```

See also: HEX\$

ON BREAK

ON BREAK GOSUB *label*

ON BREAK GOSUB 0

Tells BASIC to call the specified routine when the user presses CTRL-C or selects Stop from the Run menu.

The *label* is a label or a line number in the subroutine that receives control when the user tries to stop the program.

Example:

```
ON BREAK GOSUB 100
BREAK ON
10 GOTO 10
-
-
-
100 PRINT "Sorry, this program can't be stopped"
RETURN
```

See also: BREAK ON, Chapter 6 “Event Trapping.”

ON COLLISION

ON COLLISION GOSUB *label*

ON COLLISION GOSUB 0

Tells BASIC to call the specified routine when the COLLISION function returns a non-zero value (that is, when an object collides with the border or another object).

The *label* is a label or a line number in the subroutine that receives control. GOSUB 0 disables the COLLISION event. The ON COLLISION statement has no effect until the event has been enabled by the COLLISION ON statement.

See also: “Event Trapping” in Chapter 6, COLLISION, and OBJECT.SHAPE for an example.

ON MENU

ON MENU GOSUB *label*

ON MENU GOSUB 0

Tells BASIC to call the specified routine whenever the MENU(0) function would return a non-zero value (that is, whenever the user selects a menu item).

The *label* is a label or a line number of a subroutine to which control is passed when the MENU(0) function returns a non-zero value. GOSUB 0 disables the MENU event. The ON MENU statement has no effect until the event has been enabled by the MENU ON statement.

See also: "Event Trapping" in Chapter 6, MENU statement

ON MOUSE

ON MOUSE GOSUB *label*

ON MOUSE GOSUB 0

Tells BASIC to call the specified routine whenever the user presses the left mouse button.

The *label* is a label or line number of a subroutine to which control is passed when the user presses the left mouse button. GOSUB 0 disables the MOUSE event. The ON MOUSE statement has no effect until the event has been enabled by the MOUSE ON statement.

See also: "Event Trapping" in Chapter 6, MOUSE function, MOUSE statement.

ON TIMER

ON TIMER(*n*) GOSUB *label*

ON TIMER GOSUB 0

Tells BASIC to call the specified routine whenever a given time interval has elapsed.

The statement causes an event trap every n seconds. The *label* is a label or line number of a subroutine to which control is passed when the time interval n elapses; n must be greater than zero and less than 86400 (the number of seconds in 24 hours). GOSUB 0 disables the TIMER event.

The ON TIMER statement has no effect until the event has been enabled by the TIMER ON statement.

See also: TIMER, "Event Trapping" in Chapter 6, "Advanced Topics"

OPEN

Statement Syntax 1

```
OPEN mode, [#]filenumber, filespec[, file-buffer-size]
```

Statement Syntax 2

```
OPEN filespec[FOR mode] AS [#]filenumber[LEN=file-buffer-size]
```

Allows input or output to a disk file or device.

OPEN associates a *filenumber* with a filename.

A file must be opened before any I/O operation can be performed on that file. OPEN allocates a buffer for I/O to the disk file or device and determines the mode of access that is used with the file.

The *filenumber* is an integer expression whose value is in the range 1 to 255. The number is associated with the file for as long as it is open, and is used to refer other I/O statements to the file.

The *filespec* is a string expression containing the name of the file, optionally preceded by the name of a volume or device.

The *file-buffer-size* cannot exceed 32767 bytes. If the *file-buffer-size* option is not used, the default length is 128 bytes for random and 512 bytes for sequential files. For random files, the *file-buffer-size* should be the record length (number of characters in one record) of the file to be opened.

For sequential files, the *file-buffer-size* specification need not correspond to an individual record size, since a sequential file may have records of different sizes. When used to open a sequential file, the *file-buffer-size* specifies the number of characters to be loaded to the buffer before it is written to or read from the disk. The larger the buffer, the more room is taken from BASIC, but the faster the file I/O runs.

Syntax 1

For the first syntax, the *mode* is a string expression whose first character is one of the following:

O	Specifies sequential output mode.
I	Specifies sequential input mode.
R	Specifies random input/output mode.
A	Specifies sequential append mode.

Syntax 2

For the second syntax, the *mode* is one of the following keywords:

OUTPUT	Specifies sequential output mode.
INPUT	Specifies sequential input mode.
APPEND	Specifies sequential output mode and sets the file pointer to the end of the file. A PRINT# or WRITE# statement then adds a record to the end of the file.

If the *mode* is omitted in the second syntax, the default random access mode is assumed.

Example:

```
OPEN "ball" FOR INPUT AS 1
OPEN FileNameA$ FOR INPUT AS 2
OPEN FileNameB$ FOR OUTPUT AS 3
```

OPTION BASE

OPTION BASE *n*

Declares the minimum value for array subscripts.

This statement determines the minimum value that array subscripts may have. If *n* is 1, then 1 is the lowest value possible; if *n* is 0, then 0 is the lowest value possible. The default base is 0. Specifying an OPTION BASE other than 1 or 0 will result in a syntax error.

The OPTION BASE statement must be executed before arrays are defined or used.

Example:

If the following statement is executed, the lowest value an array subscript can have is 1.

```
OPTION BASE 1
```

PAINT

PAINT [STEP](*x,y*) [,*paintColor-id* [,*borderColor-id*]]

Paints an enclosed area the specified color.

The *x* and *y* are coordinates of any point within an area in the window containing a border—for example, any point within a circle, ellipse, or polygon.

When specified, STEP indicates that the *x* and *y* coordinates specify a pixel location *relative* to the last location referenced. When omitted, the *x* and *y* coordinates specify an *absolute* location.

The *paintColor-id* identifies the color the region is to be painted. If you omit this parameter, BASIC uses the foreground color as set by the COLOR statement.

The *borderColor-id* identifies the color of the edge of the region to be painted. If you omit this parameter, BASIC uses the color specified by *paintColor-id*.

The *paintColor-id* and *borderColor-id* are values that correspond to the *color-id* parameters in a PALETTE statements.

Note

You must specify a *type* of 16 through 31 in the WINDOW statement that created the window containing the region to be painted.

Example:

```
hue = RND*3
CIRCLE (x,y),radius,hue
PAINT (x,y),hue
```

See also: PATTERN, AREA, AREA FILL

PALETTE

PALETTE *color-id*, *red*, *green*, *blue*

Defines a "paint can" and the color it holds for reference by other BASIC statements.

The *color-id* is a value from 0 to 31 used in other BASIC statements to define a "paint can." The *depth* parameter of the SCREEN statement determines the maximum number of colors you can use, limiting the maximum value you can assign to *color-id*.

Note: The Amiga system uses *color-id* 0, 1, 2 and 3; any color assigned to these numbers through a PALETTE statement overrides the system assignments. The Amiga system initially defines color identification numbers 0, 1, 2, and 3 as follows:

0	blue
1	white
2	black
3	orange

You can reference these numbers in BASIC statements requiring a *color-id*, keeping in mind that the user can reassign colors to these numbers using the Preference Tool from the Workbench.

The *red*, *green*, and *blue* parameters each contain a value from 0.00 through 1.00 indicating a decimal percentage of red, green, and blue. Combined, these parameters define a color. The table below shows the specifications you make for *red*, *green*, and *blue* to obtain the colors indicated in the right-hand column.

Colors	Red	Green	Blue
aqua	0.00	0.93	0.87
black	0.00	0.00	0.00
blue (dark)	0.40	0.60	1.00
blue (sky)	0.47	0.87	1.00
brown	0.80	0.60	0.53
gray	0.73	0.73	0.73
green	0.33	0.87	0.00
green (lime)	0.73	1.00	0.00
orange	1.00	0.73	0.00
purple	0.80	0.00	0.93
red (cherry)	1.00	0.60	0.67
red (fire engine)	0.93	0.20	0.00
tan	1.00	0.87	0.73
violet	1.00	0.13	0.93
white	1.00	1.00	1.00
yellow	1.00	1.00	0.13

The color you specify may override previous color assignments made by the Amiga system.

Example:

```
PALETTE 1,RND,RND,RND
PALETTE 2,RND,RND,RND
COLOR 1,2
```

PATTERN

PATTERN [*line pattern*] [,*area pattern*]

Indicates the texture of text, lines, and the interior of polygons.

The *line pattern* is an integer expression that defines a 16-bit mask to be used for line drawing.

The *area pattern* is the name of an integer array containing the pattern. The array defines a 16-bit wide by N bit high mask to be used for polygon fill. In this mask, N is the number of elements in the integer array. N must be a power of two.

The values you specify for *line pattern* and *area pattern* determine the appearance of the pattern. For more information on how the values relate to the pattern drawn, see the Patterns section in the “Graphics Support Routines” chapter of the *Amiga Rom Kernel Manual*.

Example:

```
DIM AREA.PAT%(3)
AREA.PAT%(0) = &H5555
AREA.PAT%(1) = &HAAAA
AREA.PAT%(2) = &H5555
AREA.PAT%(3) = &HAAAA
PATTERN &HFFF,AREA.PAT%
```

See also: AREA and COLOR statements.

PEEKL

PEEKL(*address*)

Returns the long-integer word read from memory location (*address*).

The *address* is a numeric expression in the range from 0 to 16777216; it represents the address of the memory location. The numeric expression must be an even number; otherwise BASIC displays an error message.

The function returns the 32-bit value stored at *address*.

See also the POKE statement, which writes a long-integer word to a specified memory location.

PEEK

PEEK(*address*)

Returns a one-byte integer from memory location *address*.

The returned value is an integer in the range 0 to 255. The *address* must be in the range 0 to 16777215.

See also the POKE statement, which writes a one-byte integer to a specified memory location.

PEEKW

PEEKW(*address*)

Returns the short-integer word from memory location *address*.

The *address* is a numeric expression in the range from 0 to 16777216; it represents the address of the memory location. The numeric expression must be an even number; otherwise BASIC displays an error message.

The function returns the 16-bit value stored at *address*.

See also the POKEW statement, which writes a short-integer word to a specified memory location.

POINT

POINT (*x*,*y*)

Returns the color-id of a point in the current Output window.

The arguments *x* and *y* are the coordinates in the current Output window of the pixel to be referenced. The function returns a number that corresponds to the *color-id* in a PALETTE statement.

Coordinates (0,0) define the upper left-hand corner of the current Output window.

Coordinate values outside of the current Output window return the value -1.

POKE

POKE I, J

Writes a byte into a memory location.

I and J are integer expressions. The expression I represents the address of the memory location, and J is the data byte in the range 0 to 255. I must be in the range 0 to 16777215.

See also the PEEK statement, which returns a one-byte integer from a specified memory location.

Warning

Use POKE carefully. Altering system memory can corrupt the system. If this happens, reboot the Amiga.

See also: PEEK, VARPTR

POKEL

POKEL *address, value*

Writes a long-integer word into memory location *address*.

The *address* is a numeric expression in the range from 0 to 16777216. The numeric expression must be an even number; otherwise BASIC displays an error message.

The *value* is a numeric expression from -2147483648 to 2147483647 stored at the specified address.

See also the PEEKL statement, which returns a long-integer word from a specified memory location.

Warning

Use POKEL carefully. Altering system memory can corrupt the system. If this happens, reboot the Amiga.

POKEW

POKEW *address, value*

Writes short-integer word into memory location *address*.

The *address* is a numeric expression in the range from 0 to 16777216. The numeric expression must be an even number; otherwise BASIC displays an error message.

The *value* is a numeric expression from -65536 to 65535; numeric expressions outside this range are truncated to 16 bits and stored at the specified address.

See also the PEEKW statement, which returns a short-integer word from a specified memory location.

Warning

Use POKEW carefully. Altering system memory can corrupt the system. If this happens, reboot the Amiga.

POS

POS (*y*)

Returns the approximate line number of pen in current Output window.

The line number returned by POS is based on the width and height of the character "O" in the Output window's current font.

This value is always greater than or equal to 1. LOCATE is the inverse of the POS function.

Example:

The following example records the current line and row numbers, moves the cursor to the bottom of the screen, and prints a message; it then restores the cursor to its original position and prints a message.

```
Y = CSRLIN ' GET CURRENT CURSOR LINE NUMBER (VERTICAL POSITION)
X = POS(0) ' GET CURRENT CURSOR COLUMN NUMBER (HORIZONTAL
POSITION)
LOCATE 20,1 ' PLACE CURSOR ON LINE 20, ROW 1 (BOTTOM OF SCREEN)
PRINT "THIS PRINTS AT LOCATION 20,1 (BOTTOM OF PAGE)"
LOCATE Y,X ' PLACE CURSOR IN ORIGINAL LOCATION
PRINT "THIS PRINTS AT ORIGINAL LOCATION OF CURSOR"
```

PRESET

PRESET [STEP](*x,y*) [,*color-id*]

Sets a specified point in the current Output window.

PRESET works exactly like PSET, except that if you omit *color-id*, the specified point is set to the background color.

The *x* and *y* coordinates specify the pixel to be illuminated.

When specified, STEP indicates that the *x* and *y* coordinates specify a pixel location relative to the last location referenced. When omitted, the *x* and *y* coordinates specify an absolute location.

The *color-id* specifies the color to be used; it corresponds to the *color-id* parameter in a PALETTE statement.

If an out-of-range coordinate is given, no action is taken, and no error message is given,

The syntax of the STEP option is:

`STEP(xoffset,yoffset)`

For example, if the most recently referenced point is (10,10), then STEP (10,0) would reference a point at an offset of 10 from *x* and 0 from *y*; that is (20,10).

PRINT

PRINT [*expression-list*]

Displays data to the screen in the current Output window. (See LPRINT for information on printing data on a printer.)

If the *expression-list* is omitted, a blank line is printed. If the *expression-list* is included, the values of the expressions are printed in the Output window. The expressions in the list may be numeric or string expressions. (String constants must be enclosed in quotation marks.)

Print Positions

The position of each printed item is determined by the punctuation used to separate the items in the list. In the list of expressions, a comma causes the next value to be printed at the beginning of the next comma stop, as set by the WIDTH statement. A semicolon causes the next value to be printed

immediately adjacent to the last value. Typing one or more spaces between expressions has the same effect as typing a semicolon.

If a comma or a semicolon terminates the list of expressions, the next PRINT statement begins printing on the same line, spacing accordingly. If the list of expressions terminates without a comma or a semicolon, a carriage return is printed at the end of the line. If the printed line is longer than the line width as set by the WIDTH statement, BASIC goes to the next physical line and continues printing.

Printed numbers are always followed by a space. Positive numbers are preceded by a space. Negative numbers are preceded by a minus sign. Single precision numbers that can be represented with 7 or fewer digits in the unscaled format as accurately as they can be represented in the scaled format are output using the unscaled format. For example, 1E-7 is output as .0000001 and 1E-8 is output as 1E-08. Double precision numbers that can be represented with 16 or fewer digits in the unscaled format as accurately as they can be represented in the scaled format are output using the unscaled format. For example, 1D-15 is output as .0000000000000001 and 1D-17 is output as 1D-17.

Note

A question mark may be used in place of the word PRINT in a PRINT statement. This can be a time-saving shorthand tool, especially when entering long programs with many consecutive PRINT statements.

PRINT USING

PRINT USING *string-exp;expression-list*

Prints on the screen strings or numbers in a format you specify. (See LPRINT USING for information on printing data on a printer.)

The *string-exp* is a string literal (or variable) composed of special formatting characters. These formatting characters determine the field and the format of the printed strings or numbers. You can include literal characters in the *string-exp*. Precede with an underscore (_) each format symbol (!, &, #, etc., described later in this section) you wish to use as a literal character.

The *expression-list* contains the string expressions or numeric expressions that are to be printed; each expression must be separated by a semicolon or a comma.

String Fields

You can specify `!`, `\\`, and `&` to perform special formatting function on string fields that are to be printed.

`!` The `!` character specifies that only the first character in the string is to be printed.

`\nspaces\` `\nspaces\` represent any number of blank characters between two slashes; this specifies that 2 + *n* characters from the string are to be printed; BASIC ignores any other characters in the field. If you specify

`\\` two characters are printed, regardless of the number of characters in the field. For each space you insert between the brackets, an additional character is printed. For example,

`\ \` causes three characters to be printed. If you specify more spaces than are in the field, BASIC left-justifies the field and pads the extra spaces to the right. If you specify fewer spaces than are in the field, BASIC ignores the extra characters in the field.

`&` Specify `&` for string fields of variable length. BASIC always prints the entire string.

Numeric Fields

BASIC allows the following special characters to define the format of numeric expressions, as summarized below.

Character	Effect on Printed Output
-----------	--------------------------

#	Specifies the number of digit positions.
.	Inserts a decimal point.
+	Inserts a plus or minus sign, as applicable
-	Inserts a trailing minus sign for negative numbers.
**	Fills leading spaces with asterisks.
\$\$	Prints a dollar sign to the immediate left of a number.
**\$	Fills leading spaces with asterisks and inserts a dollar sign.
,	Prints commas where required to the left of the decimal point.
^^^	Specifies exponential format.
_	Specifies a literal character follows.

These characters are described in detail in the sections that follow. Amiga Basic treats any other character in the format string as literal output. For example,

```
PRINT USING "BALANCE = $$####.##";balance
```

#

The # character specifies the positions that must be filled with a number when the expression is printed. If the number has fewer positions than the # positions specify, BASIC justifies the number to the right and precedes it with spaces.

You can insert a decimal point within a # field; BASIC prints the # digits specified on both sides of the decimal point. BASIC precedes the decimal point with a zero if necessary.

The following examples show decimal point specifications:


```
PRINT USING "##.##";.78
PRINT USING "##.##";10.2,5.3,.234
```

The following numbers are displayed:

```
0.78
10.20 5.30 0.23
```

+

A plus sign at the beginning or end of the format string causes the sign of the number (plus or minus) to be printed before or after the number.

-

A minus sign at the end of the format field causes negative numbers to be printed with a trailing minus sign. The following examples show use of the plus and minus signs:

```
PRINT USING "+##.##";-68.95, 2.4, -9
PRINT USING "##.##-"; -68.95, 22.449, -7
```

These statements generate the following:

```
-68.95 +2.40 -9.00
68.95- 22.45 7.00-
```

**

A double asterisk at the beginning of the format string causes leading spaces in the numeric field to be filled with asterisks. The second asterisk also specifies positions for two or more digits. The statement

```
PRINT USING "***.##"; 12.39, -0.9, 765.1
```

prints the following:

```
*12.39 *-0.90 765.10
```

\$\$

A double dollar sign causes a dollar sign to be printed to the immediate left of the formatted number. The \$\$ specifies two more digit positions, one of which is the dollar sign. The exponential format cannot be used with \$\$. Negative numbers cannot be used unless the minus sign trails to the right.

The statement

```
PRINT USING "$$###.##"; 456.78, 9.3
```

prints the following:

```
$456.78 $9.30
```

**\$

The double asterisk dollar sign (**\$) at the beginning of a format string combines the effects of the two symbols. Leading spaces are filled with asterisks and a dollar sign is printed before the number. **\$ specifies three more digit positions, one of which is the dollar sign.

Do not use the exponential format with **\$. In negative numbers, minus signs appear immediately to the left of the dollar sign. The example

```
PRINT USING "***$###.##"; 2.34, 999.9
```

prints the following:

```
***$2.34*$999.90
```

If you place a comma to the left of the decimal point in a format string, BASIC prints a comma to the left of every third digit. (This has no effect on the portion of the number to the right of the decimal point.) If you place a comma at the end of the format string, BASIC prints it as part of the string. A comma specifies another digit position; it has no effect if specified with exponential (^^^) expressions. The example

```
PRINT USING "####.##"; 1234.5
PRINT USING "####.##,"; 1234.5
```

prints the following:

```
1,234.50
1234.50,
```

Place an underscore (`_`) to print the character as a literal, as shown below.

```
PRINT USING "_!##.##_!";12.34
PRINT USING "_?##.##_?";12.34
```

These statements display the following:

```
!12.34!
?12.34?
```

Place four carets (`^^^^`) after the digit position characters to specify exponential format. The four carets allow space for E+ to be printed. You can also specify a decimal point position. BASIC justifies the significant digits to the left, adjusting the exponent; unless you specify a leading + or trailing + or -, BASIC prints a space or minus sign to the left of the decimal point. The following examples shows the exponential format:

```
PRINT USING "##.##^^^^";234.56
PRINT USING ".####^^^^";888888
PRINT USING "+.##^^^^";123
```

These statements display the following:

```
2.35E+02
.8889E+06
+.12E+03
```

% Overflow Indicator

If a number is too large to fit within a field, Amiga Basic prints a % character in the result to indicate an overflow, as shown in the next example.

```
PRINT USING "##.##";987.654
```

These statement display the following:

```
%987.65
```

If the number of digits specified exceeds 24, BASIC issues the "Illegal function call" message.

PRINT#

PRINT# USING **PRINT#** *filenumber*, [**USING** *string-exp*;] *expression-list*

Writes data to a sequential file.

The *filenumber* corresponds to the number specified when the file was opened for output. The *string-exp* can consist of any of the formatting characters described under "PRINT USING." The *expression-list* are numeric or string expressions to be written to the file.

PRINT# does not compress data, but rather writes it to the file just as PRINT displays it on a screen. Therefore, be sure to delimit the data to ensure writing only the data you require in the correct format.

Delimit *numeric* expressions in *expression-list* as shown in the following example:

```
PRINT #1,A;B;C;X;Y;Z
```

(Commas used as delimiters cause extra blanks to be written to the file.)

Delimit *string* expressions with semicolons *and* special delimiters (instead of semicolons alone) so that they can be processed separately when a

program reads them in from the file using INPUT#. Here is what happens when strings are delimited with semicolons *only*:

```
A$ = "CAMERA"  
B$ = "93604 - 1"  
PRINT# 1,A$;B$
```

Both A\$ and B\$ appear as one contiguous string in the record:

```
CAMERA93604-1
```

This can be corrected by specifying a comma as a special delimiter as follows:

```
PRINT# 1,A$;", ";B$
```

which writes the following to the file:

```
CAMERA,93604-1
```

A program can process this format as two separate variables.

Surround each string that contains commas, semicolons, leading blanks, or carriage returns, with explicit quotation marks using CHR\$(34). (See the explanation of CHR\$ in this chapter for information on how this function works.)

For example, the following statements

```
A$ = "CAMERA, AUTOMATIC"  
B$ = "93604-1"  
PRINT #1,A$;B$
```

write the following image to a file:

```
CAMERA, 'AUTOMATIC93604-1
```

If you read this file with the following statement

```
INPUT #1,A$,B$
```


note that the original input is now reassigned differently:

```
A$ = "CAMERA"  
B$ = "AUTOMATIC93604-1"
```

To write the data correctly to the file, use `CHR$(34)` to specify double quotation marks as follows:

```
PRINT #1,CHR$(34);A$;CHR$(34);",",CHR$(34);B$;CHR$(34)
```

Then, the statement

```
INPUT #1, A$,B$
```

assigns the variables to the correct string as follows:

```
A$ = "CAMERA, AUTOMATIC"  
B$ = "93604-1"
```

You can also use the `PRINT#` statement with the `USING` option to control the format of the file, as shown below.

```
PRINT#1,USING"$####.##,";J;K;L
```

See also: `WRITE`

PSET

PSET [STEP] (x,y) [,color-id]

Sets a point in the current Output window.

The *x* and *y* coordinates specify the pixel that is to be colored.

When specified, `STEP` indicates that the *x* and *y* coordinates specify a pixel location relative to the last location referenced. When omitted, the *x* and *y* coordinates specify an absolute location.

The *color-id* specifies the color to be used; it corresponds to the *color-id* parameter in a `PALETTE` statement.

Example:

```
'Draw a thousand stars in random locations
FOR I = 1 TO 1000
  x = INT(RND*620)
  y = INT(RND*200)
  PSET(x,y)
NEXT I
```

See also: PRESET and COLOR

PTAB

PTAB(X)

Moves the print position to pixel X.

PTAB is similar to TAB, except that PTAB indicates the pixel position rather than the character position to advance to. If the current print position is already beyond pixel X, PTAB retreats to that pixel on the same line. Pixel 0 is the leftmost position. X must be in the range 0 to 32767. PTAB may only be used in PRINT statements.

A semicolon (;) is assumed to follow the PTAB(X) function, which means PRINT does not force a carriage return.

PUT

PUT [#] *filename* [,*record-number*]
PUT [STEP] (x,y),array [(*index*[,*index*...])][,*action-verb*]

Writes a record from a random buffer to a random access file.

Draws a screen graphics image obtained in a GET statement.

The two syntaxes shown above correspond to two different uses of the PUT statement. These are called a random file PUT and a screen PUT, respectively.

Random File PUT

For the first syntax, the *filenumber* is the number under which the file was opened. If the *record-number* is omitted, BASIC will assume the next record number (after the last PUT). The largest possible record number is 16777215; the smallest is 1.

PRINT#, PRINT# USING, and WRITE# may be used to put characters in the random file buffer before executing a PUT statement, but most often, the buffer is filled by FIELD and LSET or RSET statements.

In the case of WRITE#, Amiga Basic pads the buffer with spaces up to the carriage return. Any attempt to read or write past the end of the buffer causes a "Field overflow" error message to be generated.

Screen PUT

In the second syntax, PUT uses(*x1*, *y1*) as the pair of coordinates specifying the upper left-hand corner of the rectangular image to be placed on the screen in the current Output window.

The *array* is the name assigned to the array that holds the image. (See "GET" for a discussion of array name issues.)

The *index* allows you to PUT multiple objects in each array. This technique can be used to loop rapidly through different views of an object in succession.

The *action-verb* is one of the following: PSET, PRESET, AND, OR, XOR.

If the *action-verb* is omitted, it defaults to XOR.

The *action-verb* performs the interaction between the stored image and the one already on the screen.

Example:

```
PUT (0,0),BobArray,PSET
```

See also: GET, PRESET, PSET, PRINT, WRITE, FIELD, LSET, RSET

RANDOMIZE**RANDOMIZE** [*expression*] | [TIMER]

Reseeds the random number generator.

This statement reseeds the random number generator with the *expression*, if given, where the *expression* is either an integer between -32768 and 32767, inclusive, or where the *expression* is TIMER. If the *expression* is omitted, BASIC suspends program execution and asks for a value before randomizing, by printing:

```
Random Number Seed (-32768 to 32767)?
```

If the random number generator is not reseeded, the RND function returns the same sequence of random numbers each time the program is run. To change the sequence of random numbers every time the program is run, place a RANDOMIZE statement at the beginning of the program and change the argument with each run.

The simplest way to change a random sequence of numbers with each program run is to use RANDOMIZE TIMER. In this case, the random number seed is the number of seconds that have passed since midnight.

See also: RND

READ**READ** *variable-list*

Reads values from DATA statements and assigns them to variables.

A READ statement must always be used in conjunction with a DATA statement. READ statements assign DATA statement values to variables on

a one-to-one basis. READ statement variables may be numeric or string, and the values read must agree with the variable types specified. If they do not agree, BASIC issues the "Syntax error" message.

A single READ statement may access one or more DATA statements (they are accessed in order), or several READ statements may access the same DATA statement. If the number of variables in the *variable-list* exceeds the number of elements in the DATA statements, BASIC issues an "Out of data" error message. If the number of variables specified is fewer than the number of elements in the DATA statements, later READ statements begin reading data at the first unread element. If there are no subsequent READ statements, the extra data is ignored.

To reread DATA statements from the start, use the RESTORE statement.

```
DIM CF(19)
FOR I=1 TO 19
    READ CF(I)
    PRINT CF(I)
NEXT I
DATA 0,2,4,5,7,9,11,0,1,-1, 0,0,0,0,0,0, -12,12,0
```

See also: DATA, RESTORE

REM

REM *remark*

Allows explanatory remarks to be inserted in a program.

REM statements are not executed but appear exactly as entered when the program is listed.

REM statements may be branched into from a GOTO or GOSUB statement. Execution continues with the first executable statement after the REM statement.

Remarks may be added to the end of a line by preceding the remark with a single quotation mark instead of the REM keyword.

Warning

The DATA statement treats REM as valid data, so don't specify it in a DATA statement unless you want it considered as data.

RESTORE

RESTORE [*line*]

Allows DATA statements to be reread from a specified line.

After a RESTORE statement with no specified line number is executed, the next READ statement accesses the first item in the first DATA statement in the program. If the *line* is specified, the next READ statement accesses the first item in the specified DATA statement.

```
MainLoop:
    SOUND RESUME
    RESTORE Song
    GOSUB PlaySong
    GOTO MainLoop
    -
    -
Song:
DATA 1,3,3,3
DATA 12g>ge, 12p2de, 12p2l6g3f#g3a, 16p6gab>dcced
```

RESUME

RESUME
RESUME 0
RESUME NEXT
RESUME *line*

Continues program execution after an error recovery procedure has been performed.

Any one of the four syntaxes shown above may be used, depending upon where execution is to resume:

RESUME or RESUME 0	Execution resumes at the statement that caused the error.
RESUME NEXT	Execution resumes at the statement immediately following the one that caused the error.
RESUME <i>line</i>	Execution resumes at the <i>line</i> .

A RESUME statement that is not in an error-handling routine causes a "RESUME without error" error message to be generated.

RETURN

RETURN [*line*]

Returns execution control from a subroutine.

The *line* in the RETURN statement acts as with a GOTO. If no *line* is given, execution begins with the statement immediately following the last executed GOSUB statement.

Amiga Basic includes the RETURN *line* enhancement that lets processing resume at a line that has a number or label. Normally, the program returns to the statement immediately following the GOSUB statement when the RETURN statement is encountered. However, RETURN *line* enables the user to specify another line. This permits you more flexibility in program design. This versatile feature, however, can cause problems for untidy programmers. Assume, for example, that your program contains these fragments of a program:

```

15 MOUSE ON
10 ON MOUSE GOSUB 1000
20 FOR I = 1 TO 10

30 PRINT I
40 NEXT I
50 REM NEXT PROGRAM LINE
.
.
.
200 REM PROGRAM RESUMES HERE
.
.
.
1000 'FIRST LINE OF SUBROUTINE
.
.
.
1050 RETURN 200

```

If mouse activity takes place while the FOR...NEXT loop is executing, the subroutine is performed, but program control returns to line 200 instead of completing the FOR...NEXT loop. The original GOSUB entry is canceled by the RETURN statement, and any other GOSUB, WHILE, or FOR that was active at the time of the trap remains active. But the current FOR context also remains active, and BASIC issues the "FOR without NEXT" error message.

See also: GOSUB

RIGHT\$

RIGHT\$(X\$,I)

Returns the rightmost I characters of string X\$.

If I is greater than or equal to the number of characters in X\$, it returns X\$. If I = 0, the null string (length zero) is returned. I can range from 0 to 32767.

Example:

The following routines show the use of RIGHT\$ in extracting a field from within a string containing several fields.

```
'THIS ROUTINE EXTRACTS THE ADDRESS a: FROM STRING RECORD$
/
RECORD$ = "n:JOHN JONES ss:5349 12 99 a:3633 6TH ST WACO,TX"
LENGTH = LEN(RECORD$)           'DETERMINE LENGTH OF RECORD
OFFSET = INSTR(RECORD$,"a:")     'FIND START OF ADDRESS a:
RIGHTCHAR = LENGTH - OFFSET - 1
ADDRESS$ = RIGHT$(RECORD$,RIGHTCHAR) 'EXTRACT ADDRESS FROM RECORD$
PRINT ADDRESS$
```

The following is displayed on the screen:

```
3633 6TH ST WACO,TX
```

See also: LEFT\$, MID\$

RND

RND[(X)]

Returns a random number between 0 and 1.

RND issues the same sequence of random numbers each time a program is run unless you specify a RANDOMIZE statement.

- $X < 0$ always restarts the same sequence for any given X.
- $X > 0$ or X omitted generates the next random number in the sequence.
- $X = 0$ repeats the last number generated.

Example:

In the following example, RND produces random dimensions and screen locations for graphics images.

```
FOR I = 1 TO 40
  X = INT(RND*620)  'SET HORIZONTAL LOCATION OF CENTER
  Y = INT(RND*200)  'SET VERTICAL LOCATION OF ENTER
  RADIUS = 40*RND   'SET A RANDOM RADIUS
  CIRCLE (X,Y),RADIUS 'DRAW A CIRCLE
NEXT I
```

See also: RANDOMIZE

RSET

RSET *string-variable=string-expression*

Moves data from memory to a random file buffer in preparation for a PUT statement.

See "LSET" for a discussion of both LSET and RSET.

RUN

RUN [*line*]

RUN *filename*[,R]

Executes the program currently in memory.

If the *line* is specified, execution begins on that line. Otherwise, execution begins at the first line of the program.

With the second form of the syntax, the named file is loaded from disk into memory and run. If there is a program in memory when the command executes, a requester appears permitting the program to be saved.

In the second syntax, the *filename* must be that used when the file was saved.

RUN closes all open files and deletes the current contents of memory before loading the designated program. However, with the R option, all data files remain open.

SADD

SADD(*string expression*)

Returns the address of the first byte of data in the specified string expression.

This value is only dependable until another string allocation occurs because subsequent string allocations may cause existing strings to move in memory. SADD is typically used to pass the address of a string to a machine language program.

Avoid using VARPTR (string\$) since the format of string descriptors may change in the future.

Example:

```
CALL Prompt(SADD("How many"+CHR$(0)))
```

See also: VARPTR

SAVE

SAVE [*filename* [,A]]

SAVE [*filename* [,P]]

SAVE [*filename* [,B]]

Saves a program file.

The *filename* is a quoted string. If a filename already exists, BASIC overwrites the file. If you don't specify *filename*, BASIC prompts you for the name of the file to save.

The A option saves the file in ASCII format. If the A option is not specified, Amiga Basic saves the file in a compressed binary format that can also be specified with the B option. ASCII format takes more space on the

disk, but some programs require that files be in ASCII format. For instance, the MERGE command requires an ASCII format file. Application programs may also require ASCII format in order to read the file.

The P option protects the file by saving it in an encoded binary format. When a protected file is later RUN (or loaded with LOAD), any attempt to list or edit it will fail.

SAY

SAY "*string*",[VARPTR (*P%*(0))]

Translates a list of codes you specify into a voice delivering audible speech of any language.

The *string* contains a list of *phoneme* codes. (Phonemes are units of speech composed of the syllables and words of a spoken language.) The *mode-array*, if present, is an integer array of at least 9 elements. The specifications you make in the elements define the characteristics of the voice that is speaking. If *mode-array*, is not an integer, a type mismatch error occurs.

You can construct phoneme codes using the TRANSLATE\$ function or by following the directions given in Appendix H.

The following table gives the values you can specify in *P%* to describe the characteristics of the voice that is to speak; if you don't specify *P%* (it is optional), the default values indicated in the table are in effect.

Argument	Element #	Description
pitch		Base pitch for the voice, in hertz. Specify a value between 65 and 320. The default is 110 (normal male speaking voice).

Argument	Element #	Description
inflection	1	<p>Modulation. Choose one of two values:</p> <p>0 Inflections and emphasis of syllables (default).</p> <p>1 Monotone (robot-like).</p>
rate	2	<p>Speaking rate for the voice, in words per minute. Specify a value between 40 and 400. The default is 150.</p>
voice	3	<p>Gender. Choose one of two values:</p> <p>0 Male voice (the default)</p> <p>1 Female voice</p>
tuning	4	<p>The sampling frequency, in hertz. This element controls the changes in vocal quality. Specify a value in the range of 5000 (low and rumbly) to 28000 (high and squeaky). The default is 22200.</p>
volume	5	<p>Volume. Specify a value between 0 (no sound) and 64 (loudest). The default is 64.</p>
channel	6	<p>Channel assignment for voice output. Channels 0 and 3 go to the left audio output, and channels 1 and 2 go to the right audio output. Specify one of the code numbers from the Channel Assignment Code table that follows this table.</p> <p>The default code is 10, which assigns any available left/right pair of channels.</p>

Argument	Element #	Description
mode	7	<p>Synchronization mode. Specify either 0 or 1, as described below.</p> <ul style="list-style-type: none"> 0 Synchronous speech output. Amiga Basic waits for the completion of the current execution of SAY before processing further commands. This is the default value. 1 Asynchronous speech output. Amiga Basic begins executing the current SAY statement and then immediately resumes processing subsequent commands.
control	8	<p>Narrator device control mode. This parameter instructs Amiga Basic how to process multiple SAY statements during asynchronous speech output; that is, when Array(7)=1. Specify one of the following integers:</p> <ul style="list-style-type: none"> 0 Process normally. Amiga Basic finishes executing the first SAY statement and then executes the second one. This is the default mode. 1 Stop speech processing. Amiga Basic cancels the previous statement. 2 Override processing. Amiga Basic immediately interrupts the first SAY statement and executes the second one.

Channel Assignment Codes

Value	Channel(s)
0	0
1	1
2	2
3	3
4	0 and 1
5	0 and 2
6	3 and 1
7	3 and 2
8	either available left channel
9	either available right channel
10	either available left/right pair of channels (the default)
11	any available single channel

Example:

```
FOR J = 0 to 8: READ HOW%(J): NEXT J
TEXT$ = "dhihs ihz yohr (ahmiy5gah per5sinul kumpyuw5ter) spiy4kihnx."
SAY TEXT$,HOW%
SAY TRANSLATE$ ("Hi there, how are you?")
DATA 110,0,250,0,22200,64,10,0,0
```

See also: TRANSLATE\$

SCREEN

SCREEN CLOSE

SCREEN screen-id , width, height, depth, mode

SCREEN CLOSE screen-id

The SCREEN statement defines the dimensions of a new screen, the number of colors it can hold, and the screen resolution. SCREEN CLOSE closes the screen.

In creating the screen, SCREEN allocates private memory for a bit map.

The **SCREEN CLOSE** statement releases memory allocated to the screen identified by *screen-id*.

The *screen-id* is a number from 1 to 4 which identifies the screen; **WINDOW** statements include a corresponding *screen-id* that identifies the screen in which a window is to appear.

The *width* is the width of the screen in pixels. Specify a value from 1 through 400.

The *height* is the height of the screen in pixels. Specify a value from 1 through 640.

The *depth* is the number of bit planes associated with the screen. The value you specify (1, 2, 3, 4, or 5) determines the number of colors that can be displayed on the screen, as shown in the following table.

Value	Number of Colors
1	2
2	4
3	8
4	16
5	32

The *mode* determines the pixel width of the screen (320 pixels per horizontal line for low resolution and 640 pixels for high resolution) and whether the screen is to be *interlaced*. Normally, you specify low resolution for home television screens, and high resolution for high-resolution monochrome and RGB monitors.

An interlaced screen doubles the number of horizontal lines appearing on the screen. For example, in interlaced mode, 400 lines normally fill the screen; in non-interlaced mode, 200 lines.

The table below shows the values you can specify for *mode*, and the resulting screen produced.

Mode	Screen Produced
1	Low resolution, non-interlaced.
2	High resolution, non-interlaced.
3	Low resolution, interlaced.
4	High resolution, interlaced.

Example:

```
SCREEN 1,320,200,5,1
WINDOW 2,"Lines", (10,10)-(270,170),15,1
```

SCROLL

SCROLL *rectangle*, *delta-x*, *delta-y*

Scrolls a defined area in the current Output window.

The *rectangle* has the form (x1,y1)-(x2,y2), which specifies the bounds of the rectangle in the current Output window that is scrolled.

The *delta-x* parameter indicates the number of pixels to scroll right. If the parameter is a negative number, the rectangle scrolls left.

The *delta-y* parameter indicates the number of pixels the rectangle will scroll down. A negative value will scroll the rectangle up.

The SCROLL statement is most effective when the image to be scrolled is smaller than the defined rectangle, and the areas being affected have no background.

SGN

SGN(X)

Indicates the value of X, relative to zero.

If $X > 0$, SGN(X) returns 1.

If $X = 0$, SGN(X) returns 0.

If $X < 0$, SGN(X) returns -1.

Example:

In the following example, SGN evaluates a negative, zero, and positive value respectively.

```
PRINT SGN(-299)
PRINT SGN (499 - 499)
PRINT SGN (8722)
```

The following is displayed on the screen:

```
-1
0
1
```

SHARED

SHARED *variable-list*

Makes specified variables within a subprogram common to variables of the same name in the main program.

The *variable-list* is a list of variables, separated by commas, that are shared by the subprogram and the main program. If the variable to be shared is an array, its name must be followed by parentheses. If the value of the variable is altered within the subprogram, the value is changed for that variable in the main program, and vice versa.

The SHARED statement may only be used within a subprogram. A subprogram can have several SHARED statements for different variables, just like a program can have several DIM statements for different variables.

It is advisable to group all of one subprogram's SHARED statements at the top of the subprogram.

See also: DIM SHARED

SIN

SIN(X)

Returns the sine of X, where X is in radians.

The evaluation of this function is performed in single precision when the argument is in single precision and in double precision when the argument is in double precision.

Example:

```
PRINT "SINE OF 1 IS " SIN(1)
PRINT "SINE OF 100 IS " SIN(100)
PRINT "SINE OF 1000 IS " SIN(1000)
```

The following is displayed on the screen:

```
SINE OF 1 IS .841471
SINE OF 100 IS -.5063657
SINE OF 1000 IS .8268796
```

See also: COS, TAN

SLEEP

SLEEP

Causes a BASIC program to temporarily suspend execution until an event occurs that Amiga BASIC is interested in, such as a mouse click, key press, object collision, menu select, or a timer event.

Example:

```
LOOP:
  I$ = INKEY$
  IF I$ = "X" THEN STOP
  SLEEP
GOTO LOOP
```

SOUND

SOUND *frequency, duration* [, [*volume*][, *voice*]]

SOUND WAIT

SOUND RESUME

Produces a sound from the speaker, builds a queue of sounds, and plays a queue of sounds.

The SOUND WAIT statement causes all subsequent SOUND statements to be queued until a SOUND RESUME statement is executed. This can be used to synchronize the sounds coming from the four audio channels on the Amiga (known as *voices*.) The queue has a finite limit, so if too many SOUND statements are queued without a SOUND RESUME statement, BASIC generates an out-of-memory error.

The *frequency* can be an integer or fixed point constant of single or double precision. The minimum frequency you can specify is 20 hertz, and the maximum is 15000 hertz. If you specify a frequency of less than 20 hertz, BASIC produces a 20-hertz sound; if you specify more than 15000 hertz, BASIC produces a 15000-hertz sound.

The following tables shows four octaves of notes and their corresponding frequencies. Note that doubling the frequency produces a note one octave higher.

Note	Frequency	Note	Frequency
C	130.81	C*	523.25
D	146.83	D	587.33
E	164.81	E	659.26
F	174.61	F	698.46
G	196.00	G	783.99
A	220.00	A	880.00
B	246.94	B	977.70
C	261.63	C	1046.50
D	293.66	D	1174.70
E	329.63	E	1318.50
F	349.23	F	1396.90
G	392.00	G	1568.00
A	440.00	A	1760.00
B	493.88	B	1975.50

*Middle C

The *duration* can be any numeric expression from 0 to 77. It determines how long the sound will last. One second is represented by a duration of 18.2. Therefore, the number 18.2 as a duration argument would produce a tone that lasts one second. The maximum argument of 77 would produce a sound that lasts about 4.25 seconds.

The following table relates tempo to *duration*.

Tempo		Beats Per Minute	Duration
very slow	Larghissimo	40-60	28.13-18.75
	Largo	60-66	18.75-17.05
	Larghetto		
	Grave		
	Lento		
	Adagio	66-76	17.05-14.8
slow	Adagietto		
	Andante	76-108	14.8-10.42

medium	Andantino	108–120	10.42–9.38
	Moderato		
fast	Allegretto	120–168	9.38–6.7
	Allegro		
	Vivace		
	Veloce		
	Presto		
very fast	Prestissimo	168–208	6.7–5.41

A **SOUND** statement isn't played until the complete duration of a previous **SOUND** statement.

The *volume* can range from 0 (lowest volume) to 255 (highest volume). The default volume is 127.

The *voice* indicates which of four Amiga audio channels the sound will come from. Specify 0 or 3 for the audio channel to the left speaker and 1 or 2 for the right speaker. The default is 0.

Example:

```
SOUND 440,20,100,0
```

See also: **WAVE**

SPACES\$

SPACES\$(X)

Returns a string of spaces of length X.

The expression X is rounded to an integer and must be in the range 0 to 32767.

Example:

In the following example, SPACE\$ creates two indentation variables containing blanks; the variables force text to the appropriate indented columns when displayed with PRINT.

```
INDENT5$ = SPACE$(5)
INDENT10$ = SPACE$(10)
PRINT "Level 1 Outline Heading"
PRINT INDENT5$ "Level 2 Heading"
PRINT INDENT5$ "Level 2 Heading"
PRINT INDENT10$ "Level 3 Heading"
PRINT INDENT10$ "Level 3 Heading"
PRINT "Level 1 Heading"
END
```

The following is displayed on the screen:

```
Level 1 Outline Heading
    Level 2 Heading
    Level 2 Heading
        Level 3 Heading
        Level 3 Heading
Level 1 Heading
```

See also: SPC

SPC

SPC(I)

Generates spaces in a PRINT statement. I is the number of spaces to be skipped.

SPC can be used only with PRINT and LPRINT statements. I must be in the range 0 to 255. A semicolon (;) is assumed to follow the SPC(I) function.

Example:

```
FOR I = 1 TO 5
  PRINT SPC(I) "I AM 1 COLUMN TO THE RIGHT OF THE ABOVE LINE"
NEXT I
```

The following is displayed on the screen:

```
I AM 1 COLUMN TO THE RIGHT OF THE ABOVE LINE
 I AM 1 COLUMN TO THE RIGHT OF THE ABOVE LINE
  I AM 1 COLUMN TO THE RIGHT OF THE ABOVE LINE
   I AM 1 COLUMN TO THE RIGHT OF THE ABOVE LINE
```

See also: PTAB, SPACE\$, TAB

SQR**SQR(X)**

Returns the square root of X.

X must be ≥ 0 .

The evaluation of this function is performed in single precision when the argument is in single precision and in double precision when the argument is in double precision.

Example:

```
PRINT "VALUE          SQUARE ROOT"
FOR I = 1 TO 10
  PRINT I, SQR(I)
NEXT I
END
```

The following is displayed on the screen:

VALUE	SQUARE ROOT
1	1
2	1.414214
3	1.732051
4	2
5	2.236068
6	2.44949
7	2.645751
8	2.828427
9	3
10	3.162278

STICK

STICK(*n*)

Returns the direction of joysticks.

The *n* value determines which of two joysticks (A or B) you want direction information and on which coordinate (X or Y), as follows:

n Value	Information Returned
0	Joystick A in X direction
1	Joystick A in Y direction
2	Joystick B in X direction
3	Joystick B in Y direction

STICK returns one of the following values to indicate direction, as follows:

Value	Meaning
1	Movement is upward or to the right.
0	The stick is not engaged.
-1	Movement is downward or to the left.

STRIG

STRIG(*n*)

Returns the current status of a joystick.

This function returns the information shown in the table below depending on what you specify for *n*.

n Value	Information Returned
STRIG(0)	Returns 1 if the button on joystick A was pressed since the last time STRIG(0) was invoked. Otherwise, returns 0.
STRIG(1)	Returns 1 if the button on joystick A is currently pressed. Otherwise, returns 0.
STRIG(0)	Returns 1 if joystick B was pressed since the last time STRIG(0) was invoked. Otherwise, returns 0.
STRIG(1)	Returns 1 if the button on joystick B is currently pressed. Otherwise, returns 0.

STOP

STOP

Terminates program execution and returns to immediate mode.

STOP statements can be used anywhere in a program to terminate execution. STOP is often used for debugging.

The STOP statement does not close files.

Execution can be resumed by issuing a CONT command.

See also: CONT

STR\$

STR\$(X)

Returns a string representation of the value of X.

The string returned includes a leading space for positive numbers and a leading minus sign for negative numbers.

STR\$ is not used to convert numbers into strings for random file operations. For that purpose, use the MKI\$, MKS\$, and MKD\$ functions.

See also: VAL

STRING\$

STRING\$(I,J)

STRING\$(I,X\$)

The first syntax returns a string of length I whose characters all have ASCII code J.

The second syntax returns a string of length I whose characters are all the first character of X\$.

Example:

```
PRINT STRING$(10,"C")
PRINT STRING$(10,"#")
PRINT STRING$(10,37)
```

The following is displayed on the screen:

```
CCCCCCCCC
#####
%%%%%%%%%
```

SUB	<code>SUB <i>subprogram-name</i>[(<i>formal-parameter-list</i>)]</code>	STATIC
END SUB		END SUB
EXIT SUB		EXIT SUB

Starts, ends, and exits from a subprogram.

The *subprogram-name* can be any valid Amiga Basic identifier up to 30 characters in length. This name cannot appear in any other SUB statement.

The *formal-parameter-list* can contain two types of entries: simple variables and array variables. The subscript number that is optional after array variables should contain the number of dimensions in the array, *not* the actual dimensions of the array. Entries are separated by commas. The number of parameters is limited only by the number of characters that can fit on one logical BASIC line.

STATIC means that all the variables within the subprogram retain their values from the time control leaves the subprogram until it returns.

The body of the subprogram, the statements that make it up, occurs between the SUB and END SUB statements.

The END SUB statement marks the end of a subprogram. When the program executes END SUB, control returns to the statement following the statement that called the subprogram.

The EXIT SUB statement routes control out of the subprogram and back to the statement following the CALL subprogram statement.

Before BASIC starts executing a program, it checks all subprogram-related statements. If any errors are found, the program doesn't execute. The mistakes are not trappable with ON ERROR, nor do they have error codes. The following messages can appear in an error requester when the corresponding mistake is made:

Tried to declare a SUB within a SUB.

SUB already defined.

Missing STATIC in SUB statement.

EXIT SUB outside of a subprogram.

END SUB outside of a subprogram.

SUB without an END SUB.

SHARED outside of a subprogram.

A thorough discussion of the use and advantages of subprograms can be found in Chapter 6, "Advanced Topics."

Example:

```
SUB NextLine(win) STATIC
  SHARED iDraw,iErase
  WINDOW OUTPUT win
  DrawLine iDraw,1
  DrawLine iErase,0
END SUB
```

See also: CALL, SHARED

SWAP

SWAP variable,variable

Exchanges the values of two variables.

Any type variable may be swapped (integer, single precision, double precision, string), but the two variables must be of the same type or BASIC issues a "Type mismatch" error message.

If the second variable is not already defined when SWAP is executed, BASIC issues an "Illegal function call" error message.

Example:

```
FIRST$ = "FRED"  
LAST$ = "JONES"  
PRINT FIRST$ SPC(1) LAST$  
SWAP FIRST$,LAST$  
PRINT FIRST$ SPC(1) LAST$
```

The following is displayed on the screen:

```
FRED JONES  
JONES FRED
```

SYSTEM**SYSTEM**

Closes all open files and returns control to the Workbench.

When a SYSTEM command is executed, all open files are closed.

The same result can be achieved by selecting the Quit item from the Project menu.

When SYSTEM is executed in the program or in the Output window or from the Quit selection on the Project menu, the interpreter checks to see if the program in memory has been saved. If it hasn't been, a requester appears to prompt the user to save the program.

TAN**TAN(X)**

Returns the tangent of X where X is in radians.

The evaluation of this function is performed in single precision when the argument is in single precision and in double precision when the argument is in double precision.

Example:

```
'Tangent request program
START:
INPUT "Enter a number ", NUMBER
PRINT "Tangent of " NUMBER " is " TAN(NUMBER)
INPUT "If you have another number, enter y ", YORN$
IF YORN$ = "y" GOTO START
END
```

The following is an example of the results produced by these statements:

```
Enter a number 1.777
Tangent of 1.777 is -4.780646
If you have another number, enter y n
```

See also: COS, SIN

TIME\$

TIME\$

The function retrieves the current time.

The TIME\$ function returns an eight-character string in the form *hh:mm:ss*, where *hh* is the hour (00 through 23), *mm* is minutes (00 through 59), and *ss* is seconds (00 through 59).

Example:

The following example shows the use of TIME\$ in displaying the time of day.

```
PRINT TIME$          'PRINT CURRENT TIME IN COMPUTER
```

Here is an example of the output produced by these statement.

```
08:00:40
```

TIMER ON
TIMER OFF
TIMER STOP

TIMER ON
TIMER OFF
TIMER STOP
TIMER

The statements enable, disable, and suspend event trapping based on time.

The function retrieves the number of seconds that have elapsed since midnight.

The **TIMER ON** statement enables event trapping based on time. This allows you to alter the flow of the program based on the reading of the timer by using the **ON TIMER...GOSUB** statement.

The **TIMER OFF** statement disables **ON TIMER** event trapping. Event trapping stops until a subsequent **TIMER ON** statement is executed. The **TIMER STOP** statement suspends **TIMER** event trapping. Event trapping continues, but **BASIC** does not execute the **ON TIMER...GOSUB** statement for an event until a subsequent **TIMER ON** statement is executed.

The **TIMER** function can be used to generate a random number for the **RANDOMIZE** statement. It can also be used to time programs or parts of programs.

See also: **ON TIMER**, and "Event Trapping" in Chapter 6, "Advanced Topics."

Example:

```
ON TIMER(2) GOSUB TimeSlice 'Invoke TimeSlice every 2 seconds
TIMER ON
```

TRANSLATE\$

variable = TRANSLATE\$("*string*")

Translates English words into phonemes, from which the SAY statement can produce audible speech on the Amiga.

The *string* contains the words that are to be translated and, after execution, the *variable* contains the phoneme string. The result returned to *variable* cannot exceed 32767 characters.

Example:

```
A$ = TRANSLATE$ ("There's no place like home.")  
X% = SAY(A$)
```

See also: SAY

TRON TROFF

TRON
TROFF

Traces the execution of program statements.

The Trace On option in the Run menu is the same as the TRON statement.

As an aid in debugging, the TRON statement (executed in either immediate or program execution mode or selected from the Run menu) enables a trace flag. The currently executing statement is highlighted with a rectangle in the List window, if a List window is visible.

If there is more than one statement on a line, each statement is run and highlighted separately. The trace flag is disabled with the TROFF statement, the Trace Off menu option, or when a NEW command is executed.

UBOUND

UBOUND(*array-name* [, *dimension*])

Returns the upper bounds of the dimensions of an array.

See "LBOUND" for a discussion of both LBOUND and UBOUND.

UCASE\$

UCASE\$ (*string-expression*)

Returns a string with all alphabetic characters in upper case.

This function makes a copy of the *string-expression*, converting any lowercase letters to the corresponding uppercase letter.

The UCASE\$ function provides you with a way to compare and sort strings that have been entered with different uppercase and lowercase formats. For example, if you had a program line,

```
INPUT "Do you want to continue" ,ANSWER$,
```

the user might enter, "YES", "Yes", "yes", "Y", or "y". You could route program control in the next statement by testing the first letter of the UCASE\$ of the ANSWER\$ against "Y". This makes different affirmative responses of different users work in the program. Another use of the UCASE\$ function is when you have a form entry program.

The person or people putting in form data may not consistently use uppercase format. For example, a user might enter the names "atlanta", "AUSTIN", and "Buffalo". If a normal BASIC program to alphabetize names sorted these three, they would be ordered "AUSTIN", "Buffalo", and finally, "atlanta", because when strings are sorted they are compared based on their ASCII character numbers. The ASCII character number for "A" is lower than that for "B", but all uppercase letters come before the lowercase letters, so the character "B" comes before the character "a". If you sort based on the UCASE\$ representation of the strings, the results are alphabetically ordered.

Example:

```
a$=UCASE$(a$)
IF a$="Y" THEN Response=1
IF a$="N" THEN Response=2
IF a$="C" THEN Response=3
```

Here is another example:

```
A$ = "AUSTIN"
B$ = "atlanta"
C$ = "WaXAhachIE"
PRINT A$,B$,C$
PRINT UCASE$(A$),UCASE$(B$),UCASE$(C$)
```

Notice the difference in output, shown below, between the two PRINT statements:

AUSTIN	atlanta	WaXAhachIE
AUSTIN	ATLANTA	WAXAHACHIE

VAL

VAL(X\$)

To return the numerical value of string X\$. The VAL function also strips leading blanks, tabs, and linefeeds from the argument string.

VAL is not used to convert random file strings into numbers. For that purpose, use the CVI, CVL, CVS, and CVD functions.

See also: STR\$

VARPTR

VARPTR(*variable-name*)

Returns the address of the first byte of data identified with the *variable-name*. A value must be assigned to the *variable-name* before execution of VARPTR. Otherwise, BASIC issues an "Illegal function call" error message. Any type variable name may be used (numeric, string, array). For string variables, the address of the first byte of the string

descriptor is returned. The address returned is a number in the range 0 to 16777215. For further information, see Appendix D, "Internal Representation of Numbers."

Use `VARPTR` to obtain the address of a variable or array to be passed to an assembly language subroutine. A function call of the form `VARPTR(A(0))` is usually specified when passing an array, so that the lowest-addressed element of the array is returned.

Note

Use the `SADD` function to obtain the address of a string.

All simple variables should be assigned before calling `VARPTR` for an array element, because the addresses of the arrays change whenever a new simple variable is assigned.

`PEEK, POKE, SADD, LEN`

Example:

```
' FILL ARRAY WITH MACHINE LANGUAGE PROGRAM
DIM CODE%(50)
I = 0
INFOLOOP:
  READ A : IF A = -1 THEN MACHINEPROG:
  CODE%(I) = A: I = I + 1: GOTO INFOLOOP:
MACHINEPROG:
  X% = 10: Y% = 0
  SETYTOX=VARPTR(CODE%(0))
  CALL SETYTOX(X%,VARPTR(Y%))
  PRINT Y%
END
DATA &H4E56,&H0000,&H206E,&H0008,&H30AE,&H000C,&H4E5E
DATA &H4E75,-1
```

WAVE

WAVE *voice*, *wave-definition*

Defines the shape of a sound wave for a specified audio channel.

The WAVE statement adds versatility to the SOUND statement. By using a number array to define the shape of a sound wave to be played through the speaker, you can produce more specific types of sound. You specify a height number in each element of the array. The height numbers, when put together, define a curve; that curve is the shape of the wave.

The *voice* indicates from which of four Amiga audio channels the sound will come from. Specify 0 or 3 for the audio channel to the left speaker and 1 or 2 for the right speaker.

The *wave-definition* defines the shape sound wave for *voice*. The wave-definition can be SIN or the name of an array of integers with at least 256 elements. Each element in the array must be in the range of -128 to 127.

To save space, use the ERASE statement to delete the wave-definition array after the WAVE statement is executed.

Example:

```
DEFINT A-Z
DIM Timbre(255)
FOR I=0 TO 255
    READ Timbre(I)
NEXT I
WAVE 0,SIN
WAVE 1,Timbre
WAVE 2,Timbre
WAVE 3,Timbre
```

WHILE...WEND

WHILE *expression* [*statements*] WEND

Executes a series of statements in a loop as long as a given condition is true.

If the *expression* is true (that is, it evaluates to a non-zero value), then *statements* are executed until the WEND statement is encountered. BASIC then returns to the WHILE statement and re-evaluates the *expression*. If it is still true, the process is repeated. If it is not true, execution resumes with the statement following the WEND statement.

WHILE...WEND loops may be nested to any level. Each WEND matches the most recent previous WHILE that has not been completed with an intervening WEND. An unmatched WHILE statement causes a "WHILE without WEND" error message to be generated, and an unmatched WEND statement causes a "WEND without WHILE" error message to be generated.

Warning

Do not direct program flow into a WHILE...WEND loop without entering through the WHILE statement, as this will confuse BASIC's program flow control.

Example:

```
' THIS PROGRAM CONVERTS DECIMAL VALUES TO HEXADECIMAL
ANSWER$="Y"
WHILE (ANSWER$="Y")
  INPUT "ENTER DECIMAL NUMBER ", DECIMAL
  PRINT "HEX VALUE OF " DECIMAL "IS " HEX$(DECIMAL)
  PRINT "OCTAL VALUE OF " DECIMAL "IS " OCT$(DECIMAL)
  INPUT "DO YOU WANT TO CONVERT ANOTHER NUMBER? ", ANSWER$
  ANSWER$ = UCASE$(ANSWER$)
WEND
END
```

WIDTH

WIDTH *output-device*, [*size*] [,*print-zone*]

WIDTH #*filename*, [*size*] [,*print-zone*]

WIDTH [*size*] [,*print-zone*]

WIDTH LPRINT [*size*] [,*print-zone*]

The statement sets the printed line width and print zone width in the number of standard characters for any output device.

The *output-device* may be "SCRN:", "COM1:", or "LPT1:", and if not specified is assumed to be "SCRN:".

The integer *size* is the number of standard characters that the named output device line may contain. However, the position of the pointer or the print head, as given by the POS or LPOS function, returns to zero after position 255. In Amiga's proportionally spaced fonts, the standard width for screen characters is the equivalent of the width of any of the numerals 0 through 9.

The default line width for the screen is 255.

If the *size* is 255, the line width is "infinite"; that is, BASIC *never* inserts a carriage return character.

The *filename* is a numeric expression that is the number of the file that is to have a new width assignment.

The *print-zone* argument is the value, in standard characters, to be assigned for print zone width. Print zones are similar to tab stops, and they are forced by comma delimiters in the PRINT and LPRINT statements.

If the device is specified as "SCRN:", the line width is set at the screen. Because of proportionally spaced fonts, lines with the same number of characters may not have the same length.

If the output device is specified "LPT1:", the line width is set for the line printer. The WIDTH LPRINT syntax is an alternative way to set the printer width.

When files are first opened, they take the device width as their default width. The width of opened files may be altered by using the second WIDTH statement syntax shown above.

For detailed information on generalized device I/O, see Chapter 5, "Working With Files and Devices."

See also: LPOS, LPRINT, POS, PRINT

WINDOW WINDOW *window-id* [, [*title*] [, [*rectangle*] [, [*type*] [, *screen-id*]]]]
 WINDOW CLOSE *window-id*
 WINDOW OUTPUT *window-id*
 WINDOW(*n*)

The statements create an Output window, close an Output window, or cause the named window to become the current Output window without making it the active window (front and highlighted).

The WINDOW function returns information about the current window.

The WINDOW statement performs the following functions:

- Creates and displays a new Output window, and brings it to the front of the screen.
- Makes the window current. That is, you can use statements such as PRINT, CIRCLE, and PSET to write text and graphics to the window.

To make an existing window current, without forcing it to the front of the screen, use the WINDOW OUTPUT statement.

The *window-id* is a number from 1 to N that identifies the window. Window 1 is the Output window that appears when BASIC is started, therefore you should specify 2 or higher if you want to make a new window.

The *title* is a string expression that is displayed in the window's Title Bar, if it has a Title Bar. Window 1 displays the name of the current program or "BASIC" if no program is loaded when BASIC initializes it.

The *type* determines the options available to the user in manipulating a window using the mouse. The *type* also determines whether a window appears empty or re-displays its contents once it reappears after being covered by another window.

The following table shows the values you can use in determining *type*.

Value	Meaning
1	Window size can be changed using the mouse and Sizing Gadget in the lower right-hand side of the window.
2	Window can be moved about using the Title Bar.
4	Window can be moved from front to back of other windows using the mouse and the Back Gadget in the upper right-hand corner of the window.
8	Window can be closed using the mouse and Close Gadget in the upper left-hand corner of the window.
16	Contents of window reappear after the window has temporarily been covered by another window. BASIC reserves enough memory to remember the contents of the window.

Indicate *type* by adding two or more of the values in the above table; for example, specify 5 to indicate that the user can move the window by the Title Bar and change its size through the Sizing Gadget in the lower right-hand corner of the window. Any number from 0 through 31 is a valid *type* specification.

Note: If you specify Type 1 and Type 16 (for a total of 17) BASIC reserves enough memory for the window to grow to the full size of the screen. Otherwise, BASIC reserves only enough memory for the window

size you specify; this specification consumes a large amount of memory. If the memory available to your program is limited, avoid specifying this combination in the *type* specification.

The *rectangle* specifies the physical screen boundary coordinates of the created window. It has the form (x1,y1)–(x2,y2) where (x1,y1) is the upper-left coordinate and (x2,y2) the lower-right coordinate (relative to the screen). If no coordinates are specified, the window appears at the current default for that window (the window-id's current values). The initial defaults are for a full screen.

The *screen-id* refers to a screen created with the SCREEN statement. Specify any value from 1 through 4; the default (–1) is the Workbench screen.

WINDOW CLOSE window-id makes the named window invisible. If the current Output window is closed, the window that was most recently the current output and is still visible becomes the new Output window.

WINDOW OUTPUT window-id makes the named existing window the current output window without forcing it to the front of the screen. Statements like PRINT, CIRCLE, and PSET affect this window. This allows direct output (like text, graphics, and so forth) to a background window without changing the front window.

Programs using multiple Output windows require information about the status and size of an Output window in order to respond to different situations. The WINDOW(*n*) function (where *n* is a value from 0 through 8) provides this information; the information returned *n* is shown in the table below.

n Argument	Information returned
0	The window-id of the selected Output window.
1	The window-id of the current Output window. This is the window to which PRINT or other graphics statements send their output.

n Argument	Information returned
2	The width of the current Output window.
3	The height of the current Output window.
4	The x coordinate in the current Output window where the next character is drawn.
5	The y coordinate in the current Output window where the next character is drawn.
6	The maximum legal color for the current Output window.
7	A pointer to the INTUITION WINDOW (see the manual <i>Intuition: The Amiga User Interface</i>) record for the current Output window.
8	A pointer to the RASTPORT (see the manual <i>Intuition: The Amiga User Interface</i>) record for the current Output window.

Example:

```
WINDOW 1, "Lines", (10,10)-(270,70),15
WINDOW 2, "Polygons", (310,10)-(580,70),15
WINDOW 3, "Circles", (10,95)-(270,170),15
WINDOW OUTPUT 1
```

Note: In the above example, WINDOW 1 ("Lines") covers the Amiga Basic Output window.

WRITE

WRITE [*expression-list*]

Outputs data to the screen.

If the *expression-list* is omitted, a blank line is output. If the *expression-list* is included, the values of the expressions are output to the

screen. The expressions in the list may be numeric or string expressions. They must be separated by commas.

When the printed items are output, each item is separated from the last by a comma. Printed strings are delimited by quotation marks. After the last item in the list is printed, BASIC inserts a carriage return/linefeed sequence.

WRITE outputs numeric values without the leading spaces PRINT puts on positive numbers.

Example:

```
A = 80 : B = 90 : C$ = "The End"
WRITE A,B,C$
PRINT A,B,C$
END
```

Note the difference between the WRITE and PRINT output, shown below.

```
80,90,"The End"
80          90          The End
```

WRITE#

WRITE# *filenumber*, *expression-list*

Writes data to a sequential file.

The *filenumber* is the number under which the file was opened with the OPEN statement. The expressions in *expression-list* are string or numeric expressions. They must be separated by commas.

The difference between WRITE# and PRINT# is that WRITE# inserts commas between the items as they are written to the file and delimits strings with quotation marks. Therefore, it is not necessary to put explicit delimiters in the list. A carriage return/linefeed sequence is inserted after the last item in *expression-list* is written to the file.

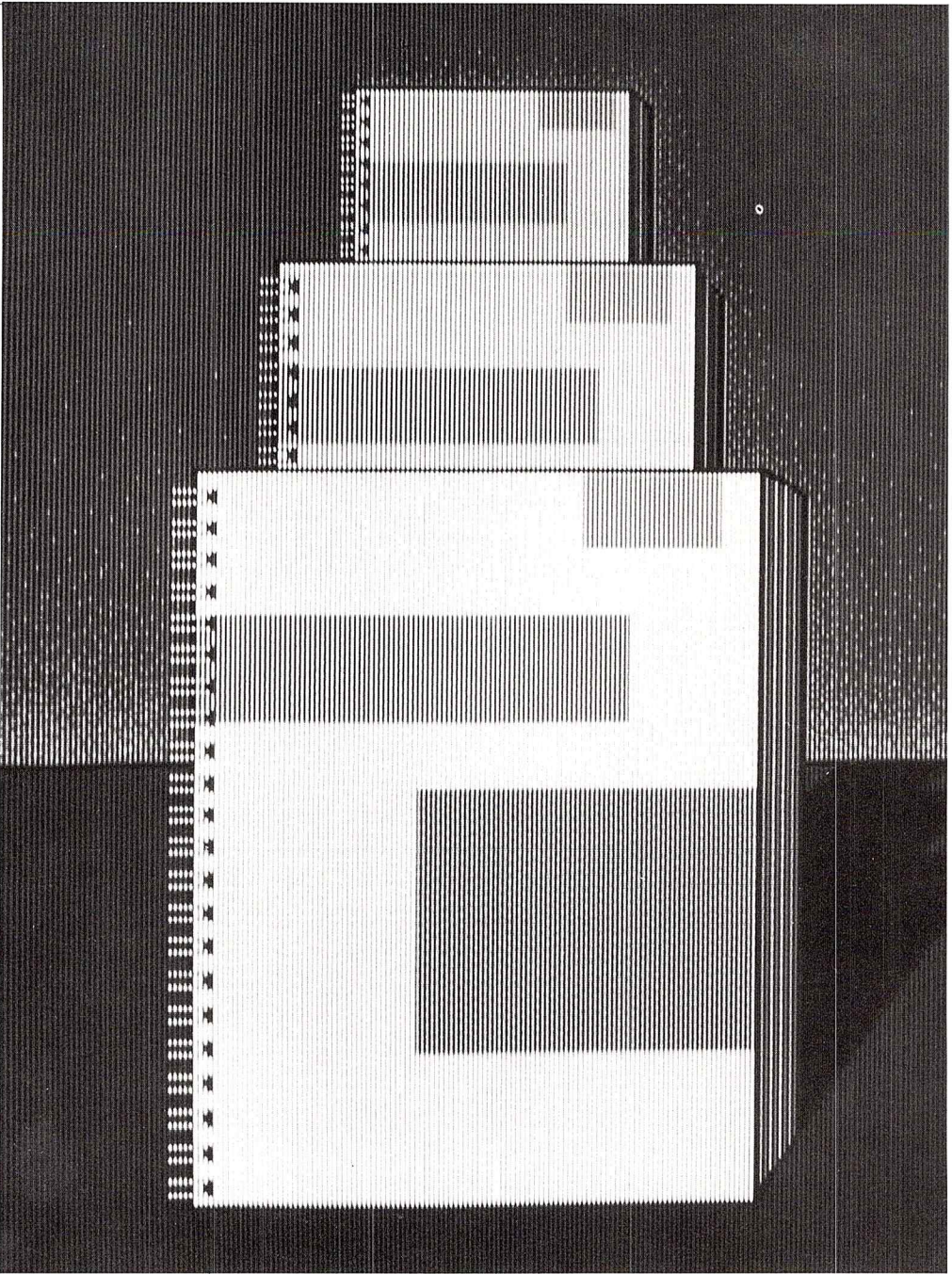
See also: OPEN, PRINT#, WRITE

Example:

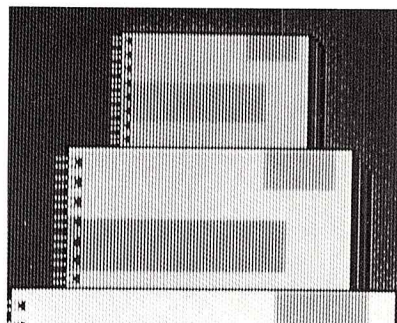
```
LET A$ = "32" : LET B = -6 : LET C$ = "Kathleen"
OPEN "O", #1, "INFO"
    WRITE #1,A$;B;C$
CLOSE #1
OPEN "I", #1, "INFO"
    INPUT #1,A$,B,C$
    PRINT A$,B,C$
CLOSE #1
END
```

This example produces the following output:

```
32          -6          Kathleen
```

Appendices



Appendix A: Character Codes

ASCII Character Codes

Dec	Hex	CHR	Dec	Hex	CHR	Dec	Hex	CHR
000	00H	NUL	043	2BH	+	086	56H	V
001	01H	SOH	044	2CH	,	087	57H	W
002	02H	STX	045	2DH	-	088	58H	X
003	03H	ETX	046	2EH	.	089	59H	Y
004	04H	EOT	047	2FH	/	090	5AH	Z
005	05H	ENQ	048	30H	0	091	5BH	[
006	06H	ACK	049	31H	1	092	5CH	\
007	07H	BEL	050	32H	2	093	5DH]
008	08H	BS	051	33H	3	094	5EH	^

Dec	Hex	CHR	Dec	Hex	CHR	Dec	Hex	CHR
009	09H	HT	052	34H	4	095	5FH	-
010	0AH	LF	053	35H	5	096	60H	'
011	0BH	VT	054	36H	6	097	61H	a
012	0CH	FF	055	37H	7	098	62H	b
013	0DH	CR	056	38H	8	099	63H	c
014	0EH	SO	057	39H	9	100	64H	d
015	0FH	SI	058	3AH	:	101	65H	e
016	10H	DLE	059	3BH	;	102	66H	f
017	11H	DC1	060	3CH	<	103	67H	g
018	12H	DC2	061	3DH	=	104	68H	h
019	13H	DC3	062	3EH	>	105	69H	i
020	14H	DC4	063	3FH	?	106	6AH	j
021	15H	NAK	064	40H	@	107	6BH	k
022	16H	SYN	065	41H	A	108	6CH	l
023	17H	ETB	066	42H	B	109	6DH	m
024	18H	CAN	067	43H	C	110	6EH	n
025	19H	EM	068	44H	D	111	6FH	o
026	1AH	SUB	069	45H	E	112	70H	p
027	1BH	ESCAPE	070	46H	F	113	71H	q
028	1CH	FS	071	47H	G	114	72H	r
029	1DH	GS	072	48H	H	115	73H	s
030	1EH	RS	073	49H	I	116	74H	t
031	1FH	US	074	4AH	J	117	75H	u
032	20H	SPACE	075	4BH	K	118	76H	v
033	21H	!	076	4CH	L	119	77H	w
034	22H	"	077	4DH	M	120	78H	x
035	23H	#	078	4EH	N	121	79H	y
036	24H	\$	079	4FH	O	122	7AH	z
037	25H	%	080	50H	P	123	7BH	{
038	26H	&	081	51H	Q	124	7CH	
039	27H	'	082	52H	R	125	7DH	}
040	28H	(083	53H	S	126	7EH	~
041	29H)	084	54H	T	127	7FH	DEL
042	2AH	*	085	55H	U			

Dec=decimal, Hex=hexadecimal(H), CHR=character, LF=LineFeed,
FF=FormFeed, CR=Carriage Return, DEL=Rubout

Non-ASCII Character Codes

b	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
b	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
b	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
b	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14
b	b	b	b												
0	0	0	0	00					SP	0	@	P	'	p	
0	0	0	1	01										NBSP	°
0	0	1	0	02										À	Đ
0	0	1	1	03										à	ð
0	1	0	0	04										Á	Ñ
0	1	0	1	05										á	ñ
0	1	1	0	06										Â	Ò
0	1	1	1	07										â	ò
1	0	0	0	08										Ã	Ó
1	0	0	1	09										ã	ó
1	0	1	0	10										Ä	Ô
1	0	1	1	11										ä	ô
1	1	0	0	12										Å	Ö
1	1	0	1	13										å	ö
1	1	1	0	14										Æ	Ø
1	1	1	1	15										æ	ø

Appendix B: Error Codes and Error Messages

Operational Errors

Error Code	Message
1	<p>NEXT WITHOUT FOR</p> <p>A variable in a NEXT statement does not correspond to any previously executed, unmatched FOR variable.</p>
2	<p>SYNTAX ERROR</p> <p>A line is encountered that contains some incorrect sequence of characters (such as an unmatched parenthesis, a misspelled command or statement, or incorrect punctuation).</p>
3	<p>RETURN WITHOUT GOSUB</p> <p>A RETURN statement is encountered for which there is no previous, unmatched GOSUB statement.</p>
4	<p>OUT OF DATA</p> <p>A READ statement is executed when there are no DATA statements with unread data remaining in the program.</p>
5	<p>ILLEGAL FUNCTION CALL</p> <p>A parameter that is out of range is passed to a math or string function. This error may also occur as the result of a negative or unreasonably large subscript.</p>

6 OVERFLOW

The result of a calculation is too large to be represented in Amiga BASIC's number format. If underflow occurs, the result is zero and execution continues without an error.

7 OUT OF MEMORY

A program is too large, has too many FOR loops or GOSUBs, too many variables, or expressions that are too complicated.

8 UNDEFINED LABEL

A line referenced in a GOTO, GOSUB, IF...THEN[...ELSE], or DELETE statement does not exist.

9 SUBSCRIPT OUT OF RANGE

Caused by one of three conditions:

1. An array element is referenced with a subscript that is outside the dimensions of the array.
2. An array element is referenced with the wrong number of subscripts.
3. A subscript is used on a variable that is not an array.

10 DUPLICATE DEFINITION

Caused by one of three conditions:

1. Two DIM statements are given for the same array.
2. A DIM statement is given for an array after the default dimension of 10 has been established for that array.
3. An OPTION BASE statement has been encountered after an array has been dimensioned by either default or a DIM statement.

11 DIVISION BY ZERO

Caused by one of two conditions:

1. A division by zero operation is encountered in an expression. Machine infinity with the sign of the numerator is supplied as the result of the division.
2. The operation of raising zero to a negative power occurs. Positive machine infinity is supplied as the result of the exponentiation, and execution continues.

12 ILLEGAL DIRECT

A statement that is illegal in immediate mode is entered as an immediate mode command. For example, DEF FN.

13 TYPE MISMATCH

A string variable name is assigned a numeric value or vice versa; a function that expects a numeric argument is given a string argument or vice versa. This error can also be caused by trying to SWAP single precision and double precision values.

14 OUT OF HEAP SPACE

The Amiga heap is out of memory. The situation may be remedied by allocating more space for the heap with the CLEAR statement. This is described in 'CLEAR' in Chapter 7, 'BASIC Reference.'

15 STRING TOO LONG

An attempt was made to create a string that exceeds 32,767 characters.

16 STRING FORMULA TOO COMPLEX

A string expression is too long or too complex. The expression should be broken into smaller expressions.

17 CAN'T CONTINUE

An attempt is made to continue a program that:

1. Has halted due to an error
2. Has been modified during a break in execution
3. Does not exist

18 UNDEFINED USER FUNCTION

A user-defined function is called before the function definition (DEF statement) is given.

19 NO RESUME

An error-handling routine is entered, but it contains no RESUME statement.

20 RESUME WITHOUT ERROR

A RESUME statement is encountered before an error-trapping routine is entered.

21 UNPRINTABLE ERROR

An error message is not available for the error condition which exists. This is usually caused by an ERROR statement with an undefined error code.

22 MISSING OPERAND

An expression contains an operator without a following operand.

23 LINE BUFFER OVERFLOW

An attempt has been made to input a line that has too many characters.

- 26 **FOR WITHOUT NEXT**
A FOR statement is encountered without a matching NEXT statement.
- 29 **WHILE WITHOUT WEND**
A WHILE statement is encountered without a matching WEND statement.
- 30 **WEND WITHOUT WHILE**
A WEND statement is encountered without a matching WHILE statement.
- 35 **UNDEFINED SUBPROGRAM**
A subprogram is called that is not in the program.
- 36 **SUBPROGRAM ALREADY IN USE**
A subprogram is called that has been previously called, but has not been ended or exited. Recursive subprograms are not permitted.
- 37 **ARGUMENT COUNT MISMATCH**
The number of arguments in a subprogram CALL statement is not the same as the number in the corresponding SUB statement.
- 38 **UNDEFINED ARRAY**
An array was referenced in a SHARED statement before it was created.
- 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, and 49 **UNPRINTABLE ERROR**
There is no error message for the error that exists.

Disk Errors

Error Code	Message
50	<p>FIELD OVERFLOW</p> <p>A FIELD statement is attempting to allocate more bytes than were specified for the record length of a random access file.</p>
51	<p>INTERNAL ERROR</p> <p>An internal malfunction has occurred in Amiga BASIC. Report to Commodore-Amiga the conditions under which the message appeared.</p>
52	<p>BAD FILE NUMBER</p> <p>A statement or command references a file with a file number that is not OPEN or is out of the range of file numbers specified at initialization.</p>
53	<p>FILE NOT FOUND</p> <p>A FILES, LOAD, NAME, or KILL command or OPEN statement references a file that does not exist on the current disk.</p>
54	<p>BAD FILE MODE</p> <p>An attempt was made to:</p> <ol style="list-style-type: none">1. Use PUT, GET, or LOF with a sequential file.2. LOAD a random access file.3. Execute an OPEN statement with a file mode other than I, O, or R.

- 55 FILE ALREADY OPEN
- A sequential output mode OPEN is issued for a file that is already open or a KILL is given for a file that is open.
- 57 DEVICE I/O ERROR
- An I/O error occurred during a disk I/O operation. It is a fatal error; that is, the operating system cannot recover from the error.
- 58 FILE ALREADY EXISTS
- The filename specified in a NAME statement is identical to a filename already in use on the disk.
- 61 DISK FULL
- All disk storage space is in use.
- 62 INPUT PAST END
- An INPUT statement is executed after all the data in the file has been INPUT, or for a null (empty) file. To avoid this error, use the EOF function to detect the end-of-file.
- 63 BAD RECORD NUMBER
- In a PUT or GET statement, the record number is either greater than the maximum allowed or equal to zero.
- 64 BAD FILE NAME
- An illegal form (for example, a filename with too many characters) is used for the filespec with a LOAD, SAVE, or KILL command or an OPEN statement.
- 67 TOO MANY OPENED FILES
- An attempt is made to create a new file (using SAVE or OPEN) when all directory entries are full.

68 DEVICE UNAVAILABLE

The device that has been specified is not available at this time.

70 PERMISSION DENIED (DISK WRITE PROTECTED)

The disk has a write protect feature, or is a disk that cannot be written to.

73 ADVANCED FEATURE

74 UNKNOWN VOLUME

A reference was made to a volume which has not been mounted.

69,71-73,75,76, 78-255 UNPRINTABLE ERROR

There is no error message for the error that exists.

Errors Reported Before Program Execution Begins

Syntax Error

A line is encountered that contains some incorrect sequence of characters (such as an unmatched parenthesis, a misspelled command or statement, or incorrect punctuation).

IF without END IF

ELSE/ ELSE IF /END IF without IF

BLOCK ELSE/END IF must be the first statement on the line

FOR without NEXT

NEXT without FOR

WHILE without WEND

WEND without WHILE

Tried to declare SUB within a SUB

SUB already defined

Missing STATIC in SUB statement

EXIT SUB outside of a subprogram

SUB without END SUB

SHARED outside of a subprogram

Statement illegal within subprogram

Too many subprograms

Appendix C: Microsoft BASIC Reserved Words

The following is a list of reserved words used in Amiga Basic. If you use these words as variable names, a syntax error will be generated.

ABS	CSRLIN	FIELD	LLIST
ALL	CVD	FILES	LOAD
AND	CVI	FIX	LOC
APPEND	CVL	FN	LOCATE
AREA	CVS	FOR	LOF
AREAFILL		FRE	LOG
AS	DATA	FUNCTION	LPOS
ASC	DATES		LPRINT
ATN	DECLARE	GET	LSET
	DEF	GOSUB	
BASE	DEFDBL	GOTO	MENU
BEEP	DEFINT		MERGE
BREAK	DEFLNG	HEX\$	MID\$
	DEFSNG		MKD\$
CALL	DEFSTR	IF	MKI\$
CDBL	DELETE	IMP	MKL\$
CHAIN	DIM	INKEY\$	MKS\$
CHDIR		INPUT	MOD
CHR\$	EDIT	INSTR	MOUSE
CINT	ELSE	INT	
CIRCLE	ELSE IF		NAME
CLEAR	END	KILL	NEW
CLNG	EOF		NEX
CLOSE	EQV	LBOUND	NOT
CLS	ERASE	LEFT\$	OBJECT.AX
COLLISION	ERL	LEN	OBJECT.AY
COLOR	ERR	LET	OBJECT.CLI
COMMON	ERROR	LIBRARY	OBJECT.CLOSE
CONT	EXIT	LINE	OBJECT.HI
COS	EXP	LIST	OBJECT.OF

OBJECT.ON	RANDOMIZE	SUB
OBJECT.PLANES	READ	SYSTEM
OBJECT.PRIORITY	REM	TAB
OBJECT.SHAPE	RESET	TAN
OBJECT.START	RESTORE	THEN
OBJECT.STOP	RESUME	TIME
OBJECT.VX	RETURN	TIMER
OBJECT.VY	RIGHT\$	TO
OBJECT.X	RND	TRANSLATE\$
OBJECT.Y	RSET	TROFFTRO
OCT\$	RUN	TRON
OFF		UBOUND
ON	SADD	UCASE\$
OPEN	SAVE	USING
OPTION	SAY	USR
OR	SCREEN	
OUTPUT	SCROLL	VA
	SGN	VARPTR
PAINT	SHARED	
PALETTE	SIN	WAIT
PATTERN	SLEEP	WAVE
PEEK	SOUND	WEND
PEEKL	SPACE\$	WHILE
PEEKW	SPC	WIDTH
POINT	SQR	WINDOW
POKE	STATIC	WRITE
POKEL	STEP	
POKEW	STICK	XOR
POS	STOP	
PRESET	STR\$	
PRINT	STRIG	
PSET	STRING\$	
PUT	SWAP	

Appendix D: Internal Representation of Numbers

Amiga Basic uses binary math. In the tables that follow, internal representation is expressed in hexadecimal numbers.

Integers in Amiga Basic

Integers are represented by a 16-bit 2's complement signed binary number.

External Representation	Internal Representation
-32768	8000
-1	FFFF
0	0000
1	0001
32767	7FFF

Binary Math

With the binary math pack, the default type for variables is single precision, and built-in mathematical functions perform in single precision or double precision. Single precision is much faster but less precise than double precision.

Double Precision

Eight bytes as follows: One bit sign followed by 11 bits of biased exponent followed by 53 bits of mantissa (including the implied leading bit which has a value of 1). If the sign bit is 0, the number is positive. If the sign bit is 1, the number is negative. The unbiased exponent (biased exponent -3FF hex or -1023 decimal) is the power of 2 by which the mantissa is to be

multiplied. The mantissa represents a number greater than or equal to 1 and less than two. Positive numbers may be represented up to but not including $1.79 * 10^{308}$. The smallest representable number is $2.23 * 10^{-308}$. Binary double precision numbers are represented with up to 15.9 digits of precision.

External Representation	Internal Representation
1	3FF0000000000000
-1	BFF0000000000000
0	000xxxxxxxxxxxxxx
10	4024000000000000
0.1	3FB9999999999999

Single Precision

Four bytes as follows: One bit sign followed by 8 bits of biased exponent followed by 24 bits of mantissa (including the implied leading bit which has a value of 1). If the sign bit is 0, the number is positive. If the sign bit is 1, the number is negative. The unbiased exponent (biased exponent -7F hex, -127 decimal) is the power of 2 by which the mantissa is to be multiplied. The mantissa represents a number greater than or equal to 1 and less than 2. Positive numbers may be represented up to but not including $3.4 * 10^{38}$. The smallest representable number is $1.18 * 10^{-38}$. Binary single precision numbers are represented with up to 7.2 digits of precision.

External Representation	Internal Representation
1	3F800000
-1	BF800000
0	00yxxxxx
10	41200000
0.1	3DCCCCCD

Appendix E: Mathematical Functions

The derived functions that are not intrinsic to Amiga Basic can be calculated as follows.

Mathematical Function	Amiga Basic Equivalent
SECANT	$\text{SEC}(X)=1/\text{COS}(X)$
COSECANT	$\text{CSC}(X)=1/\text{SIN}(X)$
COTANGENT	$\text{COT}(X)=1/\text{TAN}(X)$
INVERSE SINE	$\text{ARCSIN}(X)=\text{ATN}(X/\text{SQR}(-X*X+1))$
INVERSE COSINE	$\text{ARCCOS}(X)=-\text{ATN}(X/\text{SQR}(-X*X+1))+1.5708$
INVERSE SECANT	$\text{ARCSEC}(X)=\text{ATN}(X/\text{SQR}(X*X-1))+\text{SGN}(\text{SGN}(X)-1)*1.5708$
INVERSE COSECANT	$\text{ARCCSC}(X)=\text{ATN}(X/\text{SQR}(X*X-1))+(\text{SGN}(X)-1)*1.5708$
INVERSE COTANGENT	$\text{ARCCOT}(X)=\text{ATN}(X)+1.5708$
HYPERBOLIC SINE	$\text{SINH}(X)=(\text{EXP}(X)-\text{EXP}(-X))/2$
HYPERBOLIC COSINE	$\text{COSH}(X)=(\text{EXP}(X)+\text{EXP}(-X))/2$
HYPERBOLIC TANGENT	$\text{TANH}(X)=(\text{EXP}(-X)/\text{EXP}(X)+\text{EXP}(-X))*2+1$

Mathematical Function	Amiga Basic Equivalent
HYPERBOLIC SECANT	$\text{SECH}(X)=2/(\text{EXP}(X)+\text{EXP}(-X))$
HYPERBOLIC COSECANT	$\text{CSCH}(X)=2/(\text{EXP}(X)-\text{EXP}(-X))$
HYPERBOLIC COTANGENT	$\text{COTH}(X)=\text{EXP}(-X)/(\text{EXP}(X)-\text{EXP}(-X))^2+1$
INVERSE HYPERBOLIC SINE	$\text{ARCSINH}(X)=\text{LOG}(X+\text{SQR}(X^2+1))$
INVERSE HYPERBOLIC COSINE	$\text{ARCCOSH}(X)=\text{LOG}(X+\text{SQR}(X^2-1))$
INVERSE HYPERBOLIC TANGENT	$\text{ARCTANH}(X)=\text{LOG}((1+X)/(1-X))/2$
INVERSE HYPERBOLIC SECANT	$\text{ARCSECH}(X)=\text{LOG}((\text{SQR}(-X^2+1)+1)/X)$
INVERSE HYPERBOLIC COSECANT	$\text{ARCCSCH}(X)=\text{LOG}((\text{SGN}(X)*\text{SQR}(X^2+1)+1)/X)$
INVERSE HYPERBOLIC COTANGENT	$\text{ARCCOTH}(X)=\text{LOG}((X+1)/(X-1))/2$

Appendix F: LIBRARY FORMAT

This appendix describes the mechanism that BASIC uses to map routine names to routine offsets in the library's jump table. It is intended for the experienced programmer who needs this information to build a LIBRARY of machine language routines for BASIC. Since many routines in libraries are written in assembly language and take their arguments in registers, BASIC also requires a way to know the register calling conventions for each routine.

A special disk file that must exist for every library to be attached to BASIC with the LIBRARY statement must contain the information described above. If the library is named ":Libs/graphics.library", then this special file is named ":Libs/graphics.bmap." The .bmap extension indicates that this is a special file. The format of a ".bmap" file is as follows:

Routine name: n ASCII characters, 0-byte terminated

Offset into jump table: signed 16-bit integer,

Register parameters: n bytes terminated with a 0 byte as follows:

- 1 = pass this parameter in register d0
- 2 = pass this parameter in register d1
- 3 = pass this parameter in register d2
- 4 = pass this parameter in register d3
- 5 = pass this parameter in register d4
- 6 = pass this parameter in register d5
- 7 = pass this parameter in register d6
- 8 = pass this parameter in register d7
- 9 = pass this parameter in register a0
- 10 = pass this parameter in register a1
- 11 = pass this parameter in register a2
- 12 = pass this parameter in register a3
- 13 = pass this parameter in register a4

For routines that follow C calling conventions and take their parameters on the stack, the Register parameter is empty because BASIC doesn't need to pass any parameters in registers.

For example, if a library contained the following two routines:

```
MoveTo(x [d0], y[d1]) - library offset = -24 (decimal)
```

```
ClearRast(pRast Port [a0] - library offset = -30 (decimal)
```

then a hexadecimal dump of the library's "bmap" file would look like this:

```
4D6F7665546F00FFE8010200436C6561725261737400FFE20900
```

The utility program "ConvertFd.bas" on the Amiga Basic disk produces a .bmpa file given an .fd file as input.

Appendix G: A Sample Program

Here is a closer look at picture.bas, the program you ran in the practice session. The bracketed letters are for your reference only. They do not appear in your listing.

```
[A] DEFINT P-Z
[B] DIM P(2500)
[C] CLS
[D] LINE (0,0)-(120,120),,BF
[E] ASPECT = .1
[F]     WHILE ASPECT<20
[G]     CIRCLE(60,60),55,0,,,ASPECT
[H]     ASPECT = ASPECT*1.4
[I]     WEND
[J] GET (0,0)-(127,127),P
[K] CheckMouse:
[L]     IF MOUSE(0)=0 THEN CheckMouse
[M]     IF ABS(X-MOUSE(1)) > 2 THEN MovePicture
[N]     IF ABS (Y-MOUSE(2)) <3 THEN CheckMouse
[O] MovePicture:
[P]     PUT(X,Y),P
[Q]     X=MOUSE(1): Y=MOUSE(2)
[R]     PUT(X,Y),P
[S]     GOTO CheckMouse
```

The following section describes line by line exactly what each statement in picture.bas does.

- [A] Basic will recognize variable names beginning with the letters P through Z as integers.
- [B] Creates an array with a dimension of 2500 elements.
- [C] Erases the Output window.
- [D] Draws a rectangle defined by points (0,0) and (120,120) and filled.
- [E] Sets the variable ASPECT to 0.1.
- [F] Repeats the following as long as ASPECT is <20.
- [G] Draws an ellipse with center (60,60), radius 55, color 0 (blue), and an

aspect ratio =ASPECT.

- [H] Increases the value of ASPECT.
- [I] Exits this loop when ASPECT > = 20.
- [J] Copies the content of this part of the window to an array P.
- [K] Starts a routine called CheckMouse to check the mouse status.
- [L] Waits for the mouse Selection button to be pressed.
- [M] If the mouse has moved at least 3 points in the X direction, the program goes to MovePicture.
- [N] If the mouse has moved at least 4 points in the Y direction, the program goes to CheckMouse.
- [O] Starts a routine called MovePicture to move the picture stored in array P.
- [P] Erases the picture from the old location.
- [Q] Sets X and Y to the new coordinates of the mouse.
- [R] Copies the picture in array P to the new X,Y location.
- [S] Goes back to the CheckMouse routine.

Appendix H: Writing Phonetically for the Say Command

This appendix describes how to specify phonetic strings to the Narrator Speech synthesizer (through the SAY command). You don't need any previous experience with phonetics or with computer or foreign languages to learn this procedure. The only thing you need is a good dictionary, such as *Webster's Third International*, to look up the pronunciation of words you feel uncertain about. The beauty of writing phonetically is that you don't have to know how a word is spelled, only how it is said. Narrator lets you write down the English words that come out of your own mouth.

Narrator works on utterances at the sentence level. Even if you only want to say only one word, Narrator treats it as a complete sentence. So, Narrator asks for one of two punctuation marks to appear at the end of every sentence: the period (.) and the question mark (?). If no punctuation appears at the end of a string, Narrator automatically appends a period to it. The period, used for almost all utterances, results in a final fall in pitch at the end of the sentence.

The question mark, used only at the at the end of yes/no questions, results in a final fall in pitch. So, the question, "Do you enjoy using your Amiga?" takes a final question mark because the answer is a yes or a no. On the other hand, the question, "What is your favorite color?" doesn't take a question mark and is followed by a period. Narrator does recognize other forms of punctuation, discussed later in this appendix.

Spelling Phonetically

Utterances are usually written phonetically with an alphabet of sounds called the I.P.A. (International Phonetic Alphabet), found at the front of most good dictionaries. Since these symbols can be hard to learn and are not available on computer keyboards, the Advanced Research Projects Agency (ARPA) developed *Arpabet*, a way of representing each symbol

with one or two upper case letters. To specify phonetic sounds, Narrator uses an expanded version of Arpabet.

A phonetic sound or a phoneme is a basic speech sound, almost a speech atom. You can break sentences into words, words into syllables, and syllables into phonemes. For example, the word “cat” has three letters and (coincidentally) three phonemes. If you look at the table of phonemes, you find that three sounds make up the word cat: K, AE, and T, written as KAET. The word “cent” translates as S, EH, N, and T, or SEHNT. Note that both words begin with c, but because the c says k in cat, the phoneme k is used. You may have also noticed that there is no C phoneme. Phonetic spelling operates on a very important concept: **Spell it like it sounds—not like it looks.**

Choosing the Right Vowel

Like letters, phonemes are divided into vowels and consonants. A vowel is a continuous sound made with the vocal cords vibrating and with air exiting the mouth (rather than the nose). All vowels use a two-letter code. A consonant is any other sound, such as those made by rushing air (like S or TH) or by interruptions in air flow by the lips and the tongue (like B or T). Consonants use a one or a two-letter code.

Written English uses the five vowels a, e, i, o, and u. On the other hand, spoken English uses more than 15 vowels, and Narrator provides for most of them. To choose a vowel properly, first listen to it. Say the word aloud, perhaps extending the desired vowel sound. Then compare the sound you are making to the vowel sounds in the example words to the right of the phoneme list. For example the “a” in apple sounds the same as the “a” in cat and not like the “a’s” in Amiga, talk, or made. Note that some of the example words in the list don’t even use any of the same letters contained in the phoneme code, for example, AA as in hot.

Vowels fall into two categories: those that maintain the same sound throughout their durations and those that change their sounds. “Diphthongs” are the ones that change. You may remember being taught that vowel sounds were either long or short. Diphthongs are long vowels, but they are more complex than that. Diphthongs are the last six vowels in

the table. Say the word “made” aloud very slowly. Note how the a starts out like the e in bet but ends up like the e in beet. The a is thus a diphthong in this word and “EY” represents it. Some speech synthesizers make you specify the changing sounds in diphthongs as separate elements. Narrator assembles the diphthongal sounds for you.

Choosing the Right Consonant

Phoneticians divide consonants into many categories, but most of them are not relevant. To pick the correct consonant, you only have to pay attention to whether it is voiced or unvoiced. You make a voiced consonant with your vocal cords vibrating and you make an unvoiced one with your vocal cords silent. Written English sometimes uses the same letter combinations to represent both. Compare the sound of “th” in thin and then. Note that you make the “th” sound in thin with air rushing between the tongue and the upper teeth. In the “th” in then, the vocal cords are also vibrating. The voiced “th” phoneme is DH, the unvoiced is TH. So, the phonetic spelling of thin is THIHN whereas then is DHEHN.

A sound that is particularly subject to mistakes is voiced and unvoiced “s.” The phonetic spelling is S or Z. For example, bats ends in S while suds ends in Z. Always say words aloud to find out whether the s is voiced or unvoiced.

Another sound that causes confusion is the “r” sound. The Narrator alphabet contains two r-like phonemes: R under the consonants and ER under the vowels. If the r sound is the only sound in the syllable, use ER. Examples of words that take ER are absurd, computer, and flirt. On the other hand, if the r sound precedes or follows another vowel sound in the syllable, use R. Examples of words that take R are car, write, or craft.

Using Contractions and Special Symbols

Several of the phoneme combinations that appear in English words are created by laziness in pronunciation. For example, in the word connector, the first o is almost swallowed out of existence, so the AA phoneme is not

used and the AX phoneme is used instead. Since spoken English often relaxes vowels, AX and IX phonemes occur frequently before l, m, and n.

Narrator provides a shortcut for typing these vowel combinations. Instead of spelling “personal” PERSIXNAXL, Narrator spells it PERSINUL. Anomaly becomes UNAAMULIY instead of AXNAAMAXLIY and combination changes from KAAMBIXNEYSHIXN to KAAMBINEYSHIN. To decide whether to use the AX or IX brand of vowel relaxation, try out both and see which sounds best.

Narrator uses other special symbols internally and sometimes inserts them into your input sentence or even substitutes them for part of it. If you wish, you can type some of these symbols in directly. Probably the most useful is the Q or glottal stop— an interruption of air flow in the glottis. The word Atlantic contains one between the t and the l. Narrator already knows there should be one there and saves you the trouble of typing it. However, you may stick in a Q if Narrator should somehow let a word or a word pair slip by that would have sounded better with one.

Using Stress and Intonation

Now that you’ve learned about telling Narrator what you want said, it’s time to learn to tell it how you want it said. You use stress and intonation to alter the meaning of a sentence, to stress important words, and to specify the proper accents in words with several syllables. All this makes Narrator’s output more intelligible and natural.

To specify stress and intonation, you use stress marks made up of the single digits 1–9 followed by a vowel phoneme code. Although stress and intonation are different things, you specify them with a single number. Among other things, stress is the elongation of a syllable. So, stress is a logical term—either the syllable is stressed or it is not. To indicate stress on a given syllable, you place a number after the vowel in the syllable. Its presence indicates that Narrator is to stress the syllable. To indicate the intonation, you assign a value to the number. Intonation here means the pitch pattern or contour of an utterance.

The higher the stress marks the higher the potential for an accent in pitch. The contour of each sentence consists of a quickly rising pitch gesture up to the first stressed syllable in the sentence, followed by a slowly declining tone throughout the sentence, and finally a quick fall to the lowest pitch on the last syllable. Additional stressed syllables cause the pitch to break its slow declining pattern with rises and falls around each stressed syllable. Narrator uses a sophisticated procedure to generate natural pitch contours based on your marking of the stressed syllables.

Using Stress Marks

You place the stress marks directly to the right of the vowel phoneme codes. For example, the stress mark on the word *cat* appears after the AE, so the result is KAE5T. Generally, there is no choice about the location of the number. Either the number should go after a vowel or it shouldn't. Narrator does not flag errors such as forgetting to include a stress mark or placing it after the wrong vowel. It only tells you if a stress mark is in the wrong place, such as after a consonant. Follow these rules to use stress marks correctly:

1. Place a stress mark in a *content* word, that is, one that contains some meaning. Nouns, action verbs, and adjectives are all content words. Tonsils, remove, and huge are all examples of words that tell the listener what they're talking about. On the other hand, words like *but*, *if*, *is*, and *the* are not content words. They are, however, needed for the sentence to function and so are called *function* words.
2. Always place a stress mark on the accented syllable(s) of polysyllabic words, whether content or function. A polysyllabic word has more than one syllable. "Commodore" has its stress (or accent) on the first syllable and would be spelled KAA5MAXDOHR. "Computer" is stressed on the second syllable spelled KUMPYUW5TER. If you aren't sure about which syllable gets the stress, look the word up in a dictionary.

3. If more than one syllable in a word receives a stress mark, indicate the primary and secondary stresses by marking secondary stresses with a value of only 1 or 2. For example, the word understood has its first and last syllables stressed with stand getting primary stress and un getting secondary stress. Thus the spelling would be AH1NDERSTAE4ND.
4. Write compound words like baseball or software as one word but think of them as two words when assigning stress marks. So, spell lunchwagon as LAH5NCHWAE2GIN. Note that lunch gets a higher stress mark than wagon as the first word generally gets the primary stress.

Picking Stress Values

After you've picked the spelling and the stress mark positions correctly, it's time to decide on stress mark values. They are like parts of speech in written English. Use this table to assign stress values:

Nouns	5
Pronouns	3
Verbs	4
Adjectives	5
Adverbs	7
Quantifiers	7
Exclamations	9
Articles	0 (no stress)
Prepositions	0
Conjunctions	0
Secondary Stress	1, sometimes 2

These values only suggest a range. For example, to direct attention to a given word, you can raise its value; if you want to downplay it, lower its value. You might even want a function word to be the focus of a sentence. For example, if you assign a value of 9 to the word "to" in the sentence,

"Please deliver this to Mr. Smith," you'll indicate that the letter should be delivered to Mr. Smith in person.

Using Punctuation

In addition to periods and question marks, Narrator recognizes the dash, comma, and parentheses. The comma goes where you would normally put it in a written English sentence and tells Narrator to pause with a slightly rising pitch, indicating that there is more to come. For example, you may find that you can add more commas than you use in written English to help set off clauses from each other

The dash is like the comma except that the pitch does not rise so severely. Here's a rule of thumb: Use dashes to divide phrases and commas to divide clauses. Parentheses provide additional information to Narrator's intonation routine. Put them around noun phrases of two or more content words, for example "giant yacht." Parentheses can be particularly effective around large noun phrases like "the silliest guy I ever saw." They help provide a natural contour.

Hints for Intelligability

Although this guide should get you off to a good start, the only sure way to proficiency is to practice. Follow these tricks to improve the intelligibility of a sentence:

1. Polysyllabic words are often more recognizable than monosyllabic ones. So say enormous instead of huge. The longer version contains information in every syllable and gives the listener three times the chance to hear it correctly.

2. Keep sentences to an optimal length. Write for speaking rather than for reading. Do not write a sentence that cannot be easily spoken in one breath. Keep sentences confined to one idea.
3. Stress new terms highly the first time they are heard.

These techniques are but a few of the ways to enhance the performance of Narrator. Undoubtedly, you'll find some of your own. Have fun.

Tables of Phonemes

Vowels

Phoneme	Example	Phoneme	Example
IY	beet	IH	bit
EH	bet	AE	bat
AA	hot	AH	under
AO	talk	UH	look
ER	bird	OH	border
AX	about	IX	solid

AX and IX should never be used in stressed syllables

Diphthongs

Phoneme	Example	Phoneme	Example
EY	made	AY	hide
OY	boil	AW	power
OW	low	UW	crew

Consonants

Phoneme	Example	Phoneme	Example
R	red	L	yellow
W	away	Y	yellow
M	men	N	men
NX	sing	SH	rush
S	sail	TH	thi
F	fed	ZH	pleasure
Z	has	DH	then
V	very	J	judge
CH	check	/C	loch
/H	hole	P	put
B	but	T	toy
D	dog	G	guest
K	Commodore		

Special Symbols

Phoneme	Example	Phoneme	Example
DX (<i>tongue flap</i>)	pity	Q	kitt_en (<i>glottal stop</i>)
QX (<i>silent vowel</i>)	pause		
RX	car (<i>postvocalic R and L</i>)	LX	call
UL	=AXL	IL	=IXL
UM	=AXM	IM	=IXM
UN	=AXN (<i>contractions—see text</i>)	IN	=IXN
Digits 1–9	syllabic stress, ranging from secondary through emphatic		
.	period—sentence final character		
?	question mark—sentence final character		
–	dash—phrase delimiter		
,	comma—clause delimiter		
()	parentheses—noun phrase delimiters (see text)		

Index

- ABS, 8-21
- Amiga command key, 8-3
- animation,
 - accelerating objects, 8-89
 - bobs and sprites, 7-6, 7-8
 - causing collisions, 8-90
 - COLLISION function, 8-35, 8-37
 - creating an object, 7-2
 - confining to one area, 8-89
 - defining an object, 8-92
 - defining velocity, 8-95
 - locating object in window, 8-96
 - making object visible, 8-91
 - OBJECT statements, 8-89 - 8-96
 - prioritizing, 8-92
 - starting and stopping objects, 8-94*See also* Object Editor
- AREA, 8-21
- AREAFILL, 8-21
- arrays,
 - declaring, 6-6
 - passing elements in, 6-7
 - using LBOUND, UBOUND, 6-9
 - declaring, 8-48
 - argument expressions, 6-7
 - shared, static variables, 6-8
- ASC, 8-22
- aspect ratio,
 - definition, 8-33
 - for Amiga monitor, 8-33
 - use in drawing circles, ellipse, 8-33
- ATN, 8-23

- baud rates, Amiga, 5-2
- BEEP, 8-24
- bobs, defining, 7-6, 7-8
- BREAK,
 - command, 8-24
 - in event trapping, 6-11*See also* ON BREAK

- CALL, command description, 8-25
 - See also* subprograms
- calling programs with CHAIN, 8-28
- CDBL, 8-28
- CHAIN, 8-28
- characters, special, 8-3
- CHDIR, 8-32
- CHR\$, 8-30
- CINT, 8-31
- CIRCLE, 8-32
- CLEAR,
 - command description, 8-33
 - allocating memory with, 6-14
- CLNG, 8-38
- CLOSE, 8-34
- CLS, 8-35
- COLLISION,
 - function description, 8-35
 - Object Editor defaults, 7-2
 - in event trapping, 6-11*See also* ON COLLISION
- COLLISION ON/OFF/STOP, 8-37
- COLOR, 8-37
- colors,
 - creating, 8-103
 - determining number of, 8-134*See also* graphics commands
- COM1:, 5-2
- command key, Amiga, 8-3
- COMMON, 8-38
- concatenation, 8-18
- constants,
 - double-precision, 8-7
 - fixed-point, 8-6
 - floating-point, 8-6
 - hexadecimal, 8-6
 - integers, short and long, 8-6
 - octal, 8-6
 - single-precision, 8-7
 - types supported, 8-6
- CONT, 8-39

- Continue, 3-10
- conversion of numeric, 8-10
- Copy, 3-9
- copy key, 8-3
- COS, 8-40
- CSNG, 8-40
- CSRLIN, 8-41
- Cut, 3-9
- cut key, 8-3
- CVD, 8-42
- CVI, 8-42
- CVL, 8-42
- CVS, 8-42

- DATA, 8-42
- data segment,
 - conserving space in, 6-14
 - definition, 6-13
 - using FRE with, 6-15
- DATE\$, 8-43
- debugging programs, 4-5
- DECLARE FUNCTION, 8-44
- DEF FN, 8-45
- DEFDBL, 8-46
- DEFINT, 8-46
- DEFLNG, 8-46
- DEFSNG, 8-46
- DEFSTR, 8-46
- DELETE, 8-47
- device names, 5-1
- DIM, 8-48
- division
 - by zero, 8-14
 - integer, 8-13
- double-precision constants, 8-7

- Edit menu, 3-9
- editing a program, how to, 4-1, 2-9
- ELSE, 8-61
- END, 8-49
- END IF, 8-61
- END SUB, 8-146
 - See also* subprograms
- EOF, 8-49
- Erase, in Object Editor Tools menu, 7-6
- ERL, 8-50
- ERR, 8-50
- ERROR, 8-51
- error correction, 2-13

- event trapping,
 - activating, 6-12
 - BREAK, 6-11
 - COLLISION, 6-11
 - MENU, 6-11
 - MOUSE, 6-11
 - ON..GOSUB, 6-11
 - overview, 6-10
 - suspending, 6-12
 - terminating, 6-12
 - TIMER, 6-11
- EXIT SUB, 8-146
- exiting Amiga Basic, 3-2
- EXP, 8-52
- expressions, 8-11

- FIELD, 8-52
- filenames, valid, 5-3
- FILES, 8-53
- files,
 - opening, 5-5
 - saving, 5-5
- files, random,
 - accessing, 5-13
 - creating, 5-11
 - example, 5-15
 - overview, 5-10
- files, sequential,
 - adding data, 5-9
 - creating, 5-7
 - overview, 5-6
 - reading data from, 5-9
- FIX, 8-54
- fixed-point constants, 8-6
- floating-point constants, 8-6
- FOR, 8-54
- FRE,
 - description, 8-56
 - in memory management, 6-15
- function keys, Amiga, 8-3
- functions, types, 8-17

- GET,
 - description, 8-56
 - for random files, 8-56
 - for screen data, 8-56
- GOSUB, 8-59
- GOTO, 8-60, 8-61
- graphics commands,
 - AREA 8-21

- AREAFILL 8-21
- CIRCLE 8-32
- COLOR 8-37
- LINE 8-72
- PAINT 8-102
- PALETTE 8-103
- SCREEN 8-133

heap, *See* system heap

HEX\$, 8-60

hexadecimal constants, 8-6

high-resolution, setting, 8-134

IF..GOTO, 8-61

IF..THEN..ELSE, 8-61

immediate mode, 3-4

INKEY\$, 8-64

INPUT, 8-64

INPUT#, 8-66

INPUT\$, 8-66

INSTR, 8-67

INT, 8-68

integers,

- declaring, 8-9
- short and long, 8-6

KILL, 8-68

KYBD:, 5-2

labels,

- format and rules, 8-5

LBOUND,

- description, 8-68
- using in arrays, 6-9

LEFT\$, 8-70

LEN, 8-70

LET, 8-70

LIBRARY,

- description, 8-71
- with CALL, 8-27

LINE, 8-72

Line, in Object Editor Tools menu,

LINE INPUT, 8-72

LINE INPUT#, 8-73

LIST, command, 8-74

list key, 8-3

List window, 2-2

List window, selecting, 3-7

LLIST, 8-75

LOAD, command, 8-75

loading a program, 3-3

LOC, 8-76

LOCATE, 8-76

LOF, 8-77

LOG, 8-77

loops, nested, 8-55

low-resolution, setting, 8-134

LPOS, 8-78

LPRINT, 8-78

LPRINT USING, 8-78

LPT1:, 5-2

LSET, 8-79

machine language programs,

- calling, 8-26
- using SADD, 8-129

See also subprograms

MENU,

- description, 8-79
- in event trapping, 6-11

See also ON MENU

menu bar, displaying, 3-5

MENU ON/OFF/STOP, 8-81

menus,

- Edit, 3-9
- Project, 3-8
- Run, 3-9
- Windows, 3-11

MERGE, 8-81

MID\$, 8-82

MKD\$, 8-83

MKI\$, 8-83

MKL\$, 8-83

MKS\$, 8-83

mode, screen, 8-134

MOUSE,

- description, 8-84
- in event trapping, 6-11

See also ON MOUSE

MOUSE ON/OFF/STOP, 8-87

mouse

- position, 8-86
- status, 8-85

NAME, 8-88

NEW, 8-88

- New,
 - in Basic menu, 3-8
 - in Object Editor File menu, 7-5
- NEXT, 8-54
- Object Editor,
 - purpose, 7-2
 - how to use, 7-7
 - menus, 7-5
 - screen layout, 7-3 - 7-5
- OBJECT.AX, 8-89
- OBJECT.AY, 8-89
- OBJECT.CLIP, 8-89
- OBJECT.CLOSE, 8-91
- OBJECT.HIT, 8-90
- OBJECT.OFF, 8-91
- OBJECT.ON, 8-91
- OBJECT.PLANES, 8-92
- OBJECT.PRIORITY, 8-92
- OBJECT.SHAPE, 8-92
- OBJECT.START, 8-94
- OBJECT.STOP, 8-94
- OBJECT.VX, 8-95
- OBJECT.VY, 8-95
- OBJECT.X, 8-96
- OBJECT.Y, 8-96
- object's, how to create, 7-7 - 7-9
- OCT\$, 8-97
- octal constants, 8-6
- ON BREAK, 8-98
- ON COLLISION, 8-98
- ON MENU, 8-98
- ON MOUSE, 8-99
- ON TIMER, 8-99
- ON..GOSUB, in event trapping, 6-11
- OPEN, 8-100
- Open,
 - in Basic File menu, 3-8
 - in Object Editor File menu, 7-5
- operators,
 - arithmetic, 8-12
 - logical, 8-15
 - relational, 8-14, 8-18
- OPTION BASE, 8-102
- Output window, 2-2, 3-6
- Oval, in Object Editor Tools menu, 7-6
- overflow, 8-14

- PAINT, 8-102
- Paint,
 - in Object Editor Tools menu, 7-6
- PALETTE, 8-103
- parity, 5-2
- Paste, 3-9
- paste key, 8-3
- PATTERN, 8-105
- PEEK, 8-106
- PEEKL, 8-106
- PEEKW, 8-106
- Pen, in Object Editor Tools menu, 7-6
- POINT, 8-107
- POKE, 8-107
- POKEL, 8-108
- POKEW, 8-108
- POS, 8-109
- PRESET, 8-109
- PRINT, 8-110
- PRINT USING, 8-111
- PRINT#, 8-117
- PRINT# USING, 8-117
- printer device name, 5-3
- program mode, 3-4
- Project menu, 3-8
- PSET, 8-119
- PTAB, 8-120
- PUT,
 - description, 8-120
 - for random files, 8-120
 - for screen data, 8-120
- Quit, in Object Editor File menu, 7-5
- random
 - files, 5-10 - 5-16
 - GET, 8-56
 - PUT, 8-120
- RANDOMIZE, 8-122
- READ, 8-122
- Rectangle, in Object Editor Tools menu, 7-6
- resolution, screen, 8-134
- REM, 8-123
- RESTORE, 8-123
- RESUME, 8-123

RETURN, 8-125, 8-59
 RIGHTS\$, 8-126
 RND, 8-127
 RSET, 8-128
 RUN, command, 8-128
 Run menu, 3-9

 SADD, 8-129
 SAVE, 8-129
 Save,
 in Basic File menu, 3-8
 in Object Editor File menu, 7-5
 Save As,
 in Basic File menu, 3-8
 in Object Editor File menu, 7-5
 saving a program, 3-3, 2-20
 SAY, 8-130
 SCREEN,
 description, 8-133
 using system heap, 6-14
 SCREEN CLOSE, 8-133
 screen mode, setting, 8-134
 SCRN:, 5-2
 SCROLL, 8-135
 scrolling program listings, 4-4, 2-8
 selecting text, 4-3
 sequential files, 5-7 - 5-10
 SGN, 8-136
 SHARED, 6-5, 8-136
 Show List Window, 3-11
 Show Output Window, 3-11
 SIN, 8-137
 single-precision constants, 8-7
 SLEEP, 8-137
 SOUND,
 description, 8-138
 using system heap, 6-14
 space in stack, 8-56
 SPACES\$, 8-140
 space, determining system, 8-56
 SPC, 8-141
 speech, creating,
 with SAY, 8-130
 manually, A-23
 with TRANSLATE\$, 8-151
 sprites, defining in Object Editor, 7-6, 7-8
 SQR, 8-142

 stack,
 conserving space in, 6-13
 definition, 6-13
 using FRE with, 6-15
 Start, 3-10
 start run key, 8-3
 starting Amiga Basic, 3-2
 STATIC, 6-5
 Step option, in debugging programs, 4-6, 3-10
 STICK, 8-143
 Stop, 3-10
 STOP, 8-144
 STR\$, 8-145
 STRIG, 8-144
 STRING\$, 8-145
 strings, 8-17
 SUB, 8-156
 See also subprograms
 subprograms,
 advantages, 6-2
 calling, 8-25
 delimiting, 6-5
 referencing arrays in, 6-8
 referencing in CALL, 6-3
 shared variables in, 6-8
 static variables in, 6-8
 Suspend, 4-6, 3-10
 SWAP, 8-147
 SYSTEM, 8-148
 system heap,
 conserving space in, 6-14
 definition, 6-14
 using FRE with, 6-15

 TAN, 8-148
 THEN, 8-61
 TIME\$, 8-149
 TIMER, in event trapping, 6-11
 TIMER ON/OFF/STOP, 8-150
 Trace Off, 3-10
 Trace On, 3-10
 TRANSLATE\$, 8-151
 TROFF, 8-151
 TRON,
 description, 8-151
 in debugging programs, 4-5

UBOUND,
 description, 8-68
 using in arrays, 6-9
UCASE\$, 8-152

VAL, 8-153
variables,
 declaring, 8-9
 in arrays, 8-10
VARPTR, 8-153

WAVE,
 description, 8-155
 using system heap, 6-14

WEND, 8-156
WHILE..WEND, 8-156
WIDTH, 8-157
WINDOW,
 function, 8-160
 statement, 8-158
 using system heap, 6-14
window,
 creating, 8-158
 getting information on, 8-160
Windows menu, 3-11
WRITE, 8-161
WRITE#, 8-161

Amiga Basic: Errata for Revision A

6 December 1985

The following items are known errata to the Amiga Basic manual. This document does not reflect minor typographical errors and similar mistakes.

Note: Throughout the text, references to "BASIC" should be changed to "Amiga Basic."

The manual states that executable files have the ".bas" extension. This extension is valid, but not required with Amiga Basic. The accompanying demo programs do *not* use this extension.

Page No.	Description
1-4	<p>Second paragraph should be changed to read as follows:</p> <p>It is also true that significant Macintosh MS-BASIC tm and IBM-PC BASIC tm applications</p>
2-2	<p>Replace the bulleted items as follows:</p> <ul style="list-style-type: none">• Turn on the Amiga power switch. If the Amiga prompts you for a Kickstart diskette, then insert it in the internal drive.• Once the Workbench diskette prompt appears, put the Workbench diskette into the disk drive and wait until the Workbench icon appears and disk activity has ceased.• Put the Amiga Extras disk into any 3 1/2" Amiga disk drive.• Open the Extras disk icon. Then open the Amiga Basic icon.
2-3	<p>Second sentence should be changed to read as follows:</p> <p>To display the contents of the Extras disk in the Output window....</p>

- 2-4 First sentence under "Loading Picture" should be changed to read as follows:

Start by loading the program called Picture, which is in the BasicDemos drawer (or directory).

Second sentence of first bulleted item; delete the word "Close" from the list of menu items.

- 2-5 Second bulleted item should be changed to read as follows:

- Type
BasicDemos/Picture

After third bulleted item, add the following note:

Note: For more information on specifying directory names and file names, see the *AmigaDOS Users Manual*.

- 2-8 First paragraph under "Moving Through the List Window" should be changed to read as follows:

...lower right corner of the Amiga keyboard....

- 2-10 First sentence of second bulleted item; change to read as follows:

Release the Selection button and move the pointer to the Sizing Gadget....

- 3-2 Second bulleted item should be changed to read as follows:

- Type

AmigaBasic

on the CLI screen (selected from the System drawer)....

- 4-4 Fourth bulleted item, first sentence, should be changed to read as follows:

...the display scrolls 75% to the right.

4-7 First sentence should be changed to read as follows:

. . . you can enter the CONT command . . .

5-12 Programming example under item 3, first line, should be changed to the following:

LSET N\$ = X\$

7-2 Fourth bulleted item should be changed to read as follows:

- paint the interior of the objects with the border color

7-3 First sentence; delete the word "draw."

Last paragraph, under Menu Bar, should read as follows:

Three menus are available: File, Tools, and Enlarge. The Tools menu provides several methods of creating images. The File Menu provides a means of retrieving and saving the object files you create. The Enlarge Menu lets you expand your object for adding fine details.

7-4 Add the following paragraph at the end of the page:

To create objects with more than four colors, change the ObjEdit program (comments are included in the program listing to help you do this). The program you write that displays the resulting images must create a screen of like depth.

7-5 Delete the last sentence of first paragraph.

7-6 First sentence should be changed to read as follows:

. . . summarizes the items in the Tools menu.

Before heading entitled "A Note about Bobs and Sprites," add the following:

The following table summarizes the items in the Enlarge Menu:

Item	Function
4x4	Expands the canvas by a factor of four. The canvas size must be no larger than 100 pixels across by 31 pixels down.
1x1	Restores expanded canvas to normal size.

7-7 Change text under "How to Create Objects" to read as follows:

The Object Editor resides on the Extras disk in the BasicDemos drawer under the name ObjEdit. You open the editor and start operations just as you would any other Amiga Basic program. (Chapter 2 gives the steps to achieve this.) Then, follow the steps listed below.

Note: If you use a 256K machine, drag the Object Editor icon out of the BasicDemos window. Then close all windows and click on the Object Editor icon. This frees a maximum amount of memory for using the Object Editor. If you wish to load the ObjEdit program from within Basic, use the file name "basicdemos/objedit". Also, change the line with the LIBRARY statement from

```
LIBRARY "graphics.library"
```

to

```
LIBRARY ":basicdemos/graphics.library"
```

7-8 Last paragraph, first sentence, should be changed to read as follows:

... entirely surrounded by an outline of the same color.

8-4 Add the following note before "Label Definitions":

Note: Amiga Basic executes each line you enter sequentially, regardless of the line number you assign. You should be aware of this if you are accustomed to using another Basic that sorts the lines sequentially before execution.

- 8-5 Add the following text at the end of the "Restrictions" section:
- Warning: Line numbers are used only as labels. Amiga Basic does not sort them or remove duplicates.
- 8-12 Change third and fourth items in Operators table to the following:
- | | | |
|---|-------------------------|-----|
| * | Multiplication | X*Y |
| / | Floating Point Division | X/Y |
- 8-15 Table of logical operators, result column for the AND operator:
Change:
"X and Y" to read "X AND Y".
- 8-16 First paragraph, second sentence, should be changed to read as follows:
- Logical operators convert their operands to signed, two's complement integers
- 8-17 Third paragraph should be changed to read as follows:
- and 8 = binary 1000, so
- 8-18 Under "Relational Operators," change the list of relational operators to the following:
- = < > <> <= >=
- 8-31 Add the following note after the second paragraph of the CINT function description:
- Note: For a decimal portion that is exactly .5, if the integer portion of X is even, the function rounds down. If it is odd, the function rounds up.
- In the output for the CINT programming example, the integer "6" should be "7".

- 8-32 CIRCLE command description, fourth paragraph, last sentence, should be changed to read:
- ...CIRCLE STEP(20,15) would reference a point 30 for x and 25 for y.
- 8-33 The second programming example for CIRCLE command, fourth line should be changed to read:
- ASPECT = ASPECT*1.4
- 8-34 Programming example for CLEAR command, third line should be changed to read:
- CLEAR , ,2000
- 8-35 First sentence should be changed to read:
- The END, SYSTEM, and CLEAR statements....
- Second paragraph should be changed to read:
- See also: CLEAR, END, NEW, OPEN, STOP, SYSTEM
- 8-36 Second paragraph, last sentence should be changed to read:
- ... of the window in which the collision identified by COLLISION(0) occurred.
- 8-38 Add the following note to the end of the CLNG function description:
- Note: For a decimal portion that is exactly .5, if the integer portion of X is even, the function rounds down. If it is odd, the function rounds up.
- 8-42 Syntax format for the CVS function should be changed to:
- CVS (4-byte string)

8-53 Programming example for the FIELD statement, first sentence, should be changed to read:

... of a program that opens an existing file and fields it for three variables.

Add the following line to the programming example:

```
FIELD #2, 20 AS N$, 14 AS A$, 4 AS X$
```

Add the following note after the programming example:

See page 5-11 for a complete programming example that uses the FIELD command.

Add the following note after the last paragraph under FILES:

If *string* specifies a drive number, the statement lists all files in the current directory of the disk in that drive. See the *AmigaDOS Users Manual* for details on specifying files and their pathnames.

8-65 Fourth paragraph, first sentence, should be changed to read:

... wrong type of value (string instead of numeric, etc.) causes ...

8-68 Programming example for the INT function, output line three, should be:

– 33

8-71 Syntax format for the LIBRARY statement should have double quotes around the word “filename.”:

“filename”

8-72 Fifth paragraph, last sentence, should be changed to read:

Boxes are drawn and filled in the color given by *color-id*.

- 8-76 The mathematical expression following the second paragraph should be changed to read:

Number of Bytes Read or Written / OPEN statement Record Size
= # Returned by LOC(filename)

- 8-77 First sentence of LOG function description should be changed to read as follows:

Returns the natural (base e) logarithm of X.

- 8-79 Add the following note to the MENU statement explanation for the state argument:

When you compose a menu item which is to be checkmarked, you must leave two blank spaces ahead of the item for the checkmark to be rendered.

Fourth paragraph of the MENU statement, second and third sentences; change "20" to "19."

- 8-84 Programming example for the MK\$ functions; delete third line:

GET #2,1

- 8-88 Syntax format for the NAME statement should read as follows:

NAME "old-filename" AS "new-filename"

- 8-92 Syntax format for the OBJECT.PLANES statement should be changed to the following:

OBJECT.PLANES object-id [,plane-pick] [,plane-on-off]

- 8-94 First paragraph, first sentence, should be changed to read as follows:

. . . routine that starts up and handles collisions of the objects

Programming example, IF . . END IF structure, should be changed to the following:

IF j = -2 OR j = -4 THEN
 object bounced off left or right border

```

        OBJECT VX i,-OBJECT VX(i)
ELSE
    ^ object bounced off top or bottom border
    OBJECT VY i,-OBJECT VY(i)
END IF

```

- 8-97 First sentence of OCT\$ function description should be changed to read as follows:

Returns a string that represents the octal value of the decimal argument.

- 8-99 Add the ON ERROR GOTO, ON...GOSUB, and ON...GOTO command descriptions. (See Insert 1, attached.)

- 8-100 Last paragraph, second sentence, should be changed to read as follows:

... 128 bytes for random and sequential files.

- 8-106 First sentence should have the parentheses removed from around the word "address."

- 8-109 Programming example for the POS function should be changed to:

POS(x)

POS function description, first sentence, should be changed to read as follows:

Returns the approximate column number of pen in current....

Third paragraph, second sentence, should be changed to read as follows:

The horizontal argument of the LOCATE statement is the inverse of the POS function.

Second line of programming example should have 7 spaces deleted at the beginning of the remark. This will bring the word "POSITION" up from the next line.

- 8-114 Last sentence of description of minus sign (-) as operator should be changed to read as follows:

These statements generate the following:

Program output after minus sign (-) as operator should be changed to the following:

```
-68.95 +2.40 -9.00  
68.95-22.45 7.00-
```

Programming example, last line, for double asterisk (**) as operator should be changed to the following:

```
*12.39*-0.90765.10
```

- 8-116 First paragraph, last sentence, of description of four carets (^^^) as operator should be changed to read as follows:

The following examples show the exponential format:

- 8-117 Second paragraph, last sentence, of PRINT# statement description should be changed to read as follows:

The *expression-list* items are numeric or string expressions to be written to the file.

- 8-126 Next to last paragraph of RETURN command description, last sentence, should be changed to read:

Using a RETURN from within a FOR loop is not good programming practice and should be discouraged.

- 8-130 The syntax format for the SAY command should be changed to the following:

```
SAY "string" [,mode-array]
```

Last paragraph before table; replace all occurrences of "P%" with

“mode-array”.

- 8-130 Bottom of page: Element # column for “pitch” parameter should have a zero (0).
- 8-133 The example for SAY should show that all text in the TEXT\$ statement must be in capital letters.
- 8-134 Third paragraph, second sentence; change “400 to “640”.
- Fourth paragraph, second sentence; change “640” to “400”.
- 8-138 Last paragraph, first sentence, should be changed to read as follows:
- The following table shows four octaves of notes and their . . .
- 8-148 Add the TAB function description. (See Insert 2, attached.)
- 8-151 Programming example, second line, for the TRANSLATE\$ function should be changed to read:
- SAY (A\$)
- 8-153 First sentence of the VAL statement description should be changed to read as follows:
- Returns the numeric value of string X\$.
- 8-163 Programming example, third line, for the WRITE# statement should be changed to read:
- WRITE #1,A\$,B\$,C\$

ADDITIONAL NOTES

When a file name is contained in a command, it must be enclosed within double quotes.

The following are known bugs in version 1.0 of Amiga Basic:

Using the BF option on the LINE statement (which floodfills interior of box) causes Amiga Basic to fail if the box's lower border is 200 or greater.

If a user scrolls through the List window after closing the Output window and then closes the List window, Basic returns control to Workbench but does not release the memory it used (about 128 k).

The second, ending, label or line number in the LIST syntax is ignored. For example,

```
LIST start - init
```

lists entire program from "start" onwards.

The following problems remain in the line editor:

When user types past the right border of the List window, causing an automatic rightward shift, characters are sometimes missorted.

The "Copy" feature in the Edit menu has garbage characters in its clipboard. When user highlights past the exact characters to be copied, then pastes, the result is the desired characters plus garbage. For example, if the following is highlighted and Copied:

```
xyz
```

When copied (or Cut) characters are Pasted, the result might be

```
xyz RINT "j
```

The DEL key is not implemented with respect to the line editor (although BACKSPACE does work).

The KILL and NAME commands don't affect related .INFO files, even though these are automatically generated with the SAVE command.

The following problems remain in the ObjEdit program:

Amiga Basic will fail if the user attempts to run a version of ObjEdit with a custom screen with a depth of three or more.

The dependence on user's pressing any key (using INKEY\$), in response to the 'Y or X too large' message regarding the Enlarge feature, results in confusion, since the program won't respond to menu selections until a key has been pressed (even though it appears to).

If user inputs to Enlarged canvas while it is being printed, a 16x16 blowup of rightmost portion of object prints across screen and menus cease to respond.

Comment line instructing user in creating objects in 3 bit planes should be changed to read:

"...alter the next 4 lines."

Insert 1

ON ERROR GOTO

ON ERROR GOTO *label*

Sends program control to an error-handling routine.

After enabling error handling, all errors detected cause a jump to the specified error-handling routine. If *label* doesn't exist, Amiga Basic displays an "Undefined label" error message. The RESUME statement is required to continue program execution.

To disable error handling, execute an ON ERROR GOTO 0. Subsequent errors generate an error message and halt execution. An ON ERROR GOTO 0 statement that appears in an error-handling routine causes Amiga Basic to stop and print the error message for the error that caused the trap.

It is recommended that all error-handling routines execute an ON ERROR GOTO 0 if an error is encountered for which there is no recovery action.

See also RESUME.

Example:

```
10 ON ERROR GOTO 900
80 PRINT = "Return from error"
900 IF (ERR = 230) AND (ERL = 90) THEN PRINT "try again" : RESUME 80

80 PRINT = "Return from error"
```

ON...GOSUB ON...GOTO

ON *expression* GOSUB *label-list*
• ON *expression* GOTO *label-list*

Branches to one of several specified line numbers or labels, depending on the value returned when an expression is evaluated. This is called a “computed GOSUB” or “computed GOTO.”

The value of *expression* determines which line number or label in the *label-list* is used for branching. If the value is a noninteger, the fractional portion is rounded.

The *label-list* is a series of line numbers or labels to which program control will be routed depending on the value of the expression. For example, if the value of the expression is three, the third item in the *label-list* is the destination of the branch.

In the ON...GOSUB statement, each line named in the list must be the first line of a subroutine.

If the value of the *expression* is zero, or greater than the number of items in the list (but less than or equal to 255), Amiga Basic continues with the next executable statement. If the value of the *expression* is negative or greater than 255, an “illegal function call” error message is generated.

Example:

```
'This program illustrates the use of the
'ON...GOSUB Statement
START:
INPUT "Enter your choice number (1 . . . 3) ? ",CHOICE%
IF CHOICE% < 1 OR CHOICE% > 3 THEN GOTO START:
ON CHOICE% GOSUB SUB1,SUB2,SUB3
END
SUB1:
```

Insert 2

TAB

TAB(I)

Moves the print position to I.

If the current print position is already beyond space I, TAB goes to that position on the next line. Space 1 is the leftmost position, and the rightmost position is the width. TAB may only be used in PRINT and LPRINT statements. A semicolon (;) is assumed to precede and to follow the TAB(I) function.

Example:

```
PRINT " Name";TAB(16);"Amount Due"
PRINT TAB(2);"----";TAB(16);"-----"
FOR I% = 1 to 6
    READ A$,B
    PRINT " ";A$;TAB(18);B
NEXT I%
DATA "G. T. Jones",25,"T. Bear",1
DATA "B. Charlton",33,"B.Moore",99
DATA "G. Best", 100, "N. Styles",13.50
```

These statements display the following:

Name	Amount Due
----	-----

G.T. Jones	25
T. Bear	1
B. Charlton	33
B. Moore	99
G. Best	100
N. Styles	13.5



Commodore Business Machines, Inc.
1200 Wilson Drive, West Chester, PA 19380

Commodore Business Machines, Limited
3370 Pharmacy Avenue, Agincourt, Ontario, M1W 2K4

Copyright 1985 © Commodore-Amiga, Inc.

Copyright © Microsoft® Corporation, 1985. All rights reserved.