The Atari® ST for Beginners

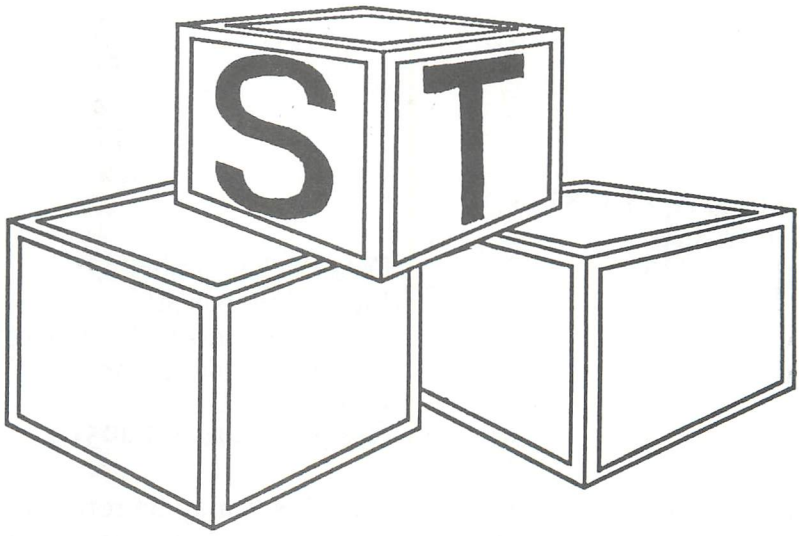
Everyone's guide to using the ST

you can count on Abacus

A Data Becker Book

The Atari® ST™ for Beginners

Ranier Lüers • Michael Stein



A Data Becker Book

Published by

Abacus

First Printing, April 1987
Printed in U.S.A.

Data Becker GmbH
Merowingerstr.30
4000 Düsseldorf, West Germany

Abacus Software, Inc.
P.O. Box 7219
Grand Rapids, MI 49510

Every effort has been made to insure complete and accurate information concerning the material presented in this book. However, Abacus Software can neither guarantee nor be held legally responsible for any mistakes in printing or faulty instructions contained in this book. The authors always appreciate receiving notice of any errors or misprints.

ISBN 0-916439-55-0

Table of Contents

Introduction

Dear ST User:

We're certain that you're on your way to becoming a genuine computer expert.

After all, you are now reading the book that will familiarize you with one of the fastest, most efficient microcomputers on the market today: the Atari ST. This is certainly enough reason to call you a future expert! You'll see that it won't take long for you to master your ST.

We'll be using a number of words and phrases that may be unfamiliar to you. These words will be printed in *italics* the first time we mention them. First-time computer users may wish to read Appendix C, a glossary for computer users, to familiarize themselves with computer terminology. You should also keep your ST Owner's Manual handy to look up these terms as needed.

After reading this book, you'll be able to program your computer in both BASIC and LOGO, and you'll understand the internal workings of your computer. We chose to explore the ST in a way that would make the new user most comfortable with it, rather than give you a typical, scientific, and throughly boring write-up of the ST. When you thirst for additional knowledge after reading this text, you can get more detailed information from other books and programs in the Atari ST Reference Library from Abacus.

This book consists of five chapters. Chapters 1 and 2 introduce you to the ST. Chapters 3 and 4 introduce you to the programming languages BASIC and LOGO. For further information, Chapter 5 holds numerous additional facts about the ST.

Are you ready? Then enough lecturing—let's look at the Atari ST.

Ranier Lüers
Michael Stein

Münster/Bochum
West Germany
1986

Acknowledgments

Our special thanks to Dr. Riedel and Atari, GmbH (Germany) who have given us a great deal of support while working on this book. Furthermore, many Atari users throughout Germany advised us. We would like to mention these names in particular:

Julian Reschke
Harald Mieling

We would also like to thank the following people for their help with the photographs and artwork in this book:

Daniel Stoffregen
Olaf Schellenberger
Ronald Goergen

Also, we would like to thank DATA WELT magazine for generously supplying us with information and photographic materials.

Chapter 1

First steps: Setting up & connecting the ST

First steps: Setting up and connecting the ST

You're probably anxious to get your ST up and running, and to start to work with some of its advanced features right away. We know from our own experience that the first steps taken in computing are usually the ones where the most frustrating mistakes occur. To minimize these problems, we'll begin this chapter by showing you how to connect the components of your ST, thereby avoiding some potentially expensive errors.

There are currently two different models of the Atari ST. The first, the 520ST, can be purchased as a set of components: the computer and its power supply, a color or monochrome monitor, one or two disk drives, and a mouse. The second model, the 1040ST, comes complete with a built-in disk drive and built-in power supply. All you need is a monitor and a mouse.

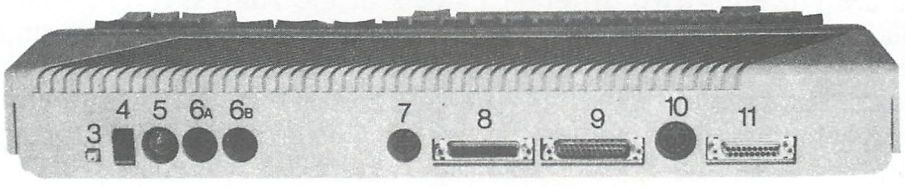
NOTE: A third ST model is available in Europe. Called the 260ST, this computer is identical to the 520ST in every respect except memory capacity. Therefore, any reference to the 520ST also applies to the 260ST.

Once your ST is assembled, it should look something like this:

The 520 ST computer system

Our next step is to connect the components of the ST. Since there are differences between the 520ST and 1040ST, we'll discuss the procedure for each model as we go along. **Do not plug in the 520ST power supply or the 1040ST power cord (or the monitor's power cord) until you complete the steps for connection outlined below.**

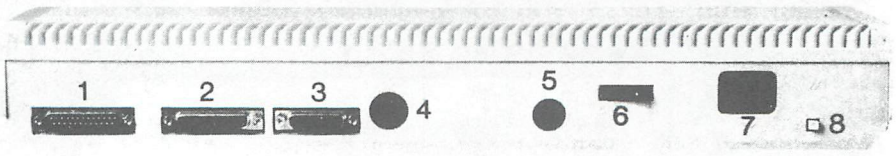
1.1 The computer



520ST—rear view

We've numbered the connections on the back of the 520ST so that they are clearly recognizable. The power supply connector cord is plugged into socket 5. Make certain that the small alignment notch of the power supply plug points upward when you plug it into the ST. **Note**: Newer 520ST's will have an additional jack between 6B and 7. This standard RCA jack is connected to an RF modulator, and lets you use a television set instead of a monitor with the 520ST.

The 1040ST's connections are a little simpler:



1040ST—rear view

The power cord for connecting the 1040ST to standard electrical current is plugged into jack 7 of the computer. There is no external power supply like with the 520ST.

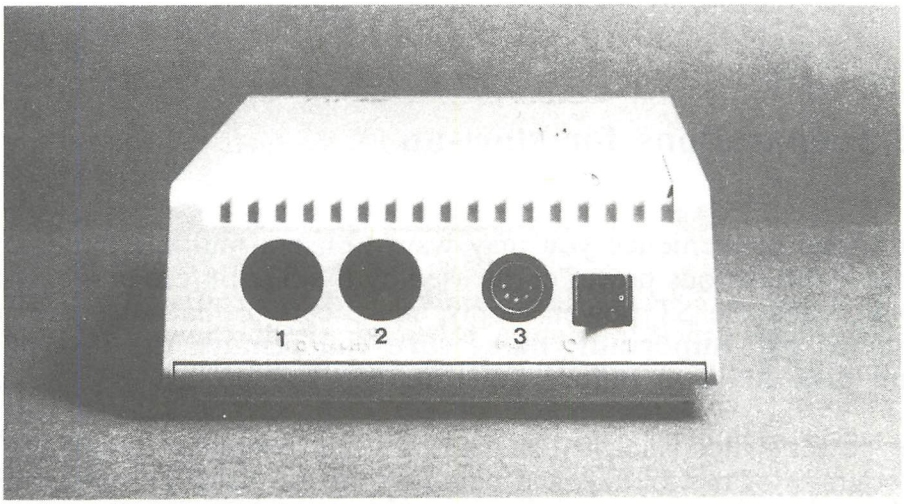
1.2 The monitor

The connector cable of the monitor is plugged into the monitor at its right rear. The cable that's attached to the monitor plugs into socket 7 of the 520ST computer. The monitor cable plugs into socket 5 of the 1040ST. As with the power connector, the small alignment notch on this cable must point upward when plugged in.

1.3 The disk drive

The disk drive has to be connected to the ST and to its own power supply. Plug the power supply cord into connection 3 of the disk drive, with the alignment notch upward. Plug the grey cable that came with the disk drive into connection 1 of the disk drive. Plug the other end into connection 10 of the 520ST. If you're a 1040ST owner, you already have an internal disk drive. If you are using second disk drive, plug it into connection 4 of the 1040ST. If you only need one drive, go directly to Section 1.4.

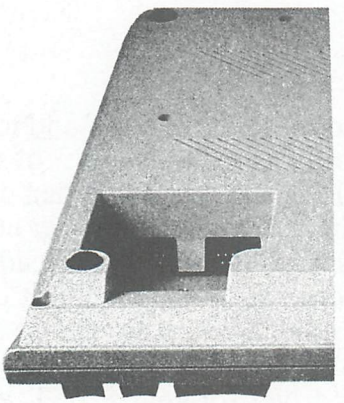
If you purchased two disk drives for your 520ST, connect the first drive as above, then connect the second disk drive to the power supply. Plug the grey cable supplied with the second disk drive into connection 2 of the first disk drive. Plug the other end into connection 1 of the second disk drive.



Disk drive—rear view

1.4 The mouse

Now we'll connect the *mouse*, that palm-sized input device with the short cord. There are two jacks on the right side of the 520ST. Connect the mouse to the port marked 0. 1040ST users must tilt the computer up, then plug the mouse into port 0 (this port is recessed in the bottom of the computer). The illustration below shows ports 0 and 1 on the 1040ST:



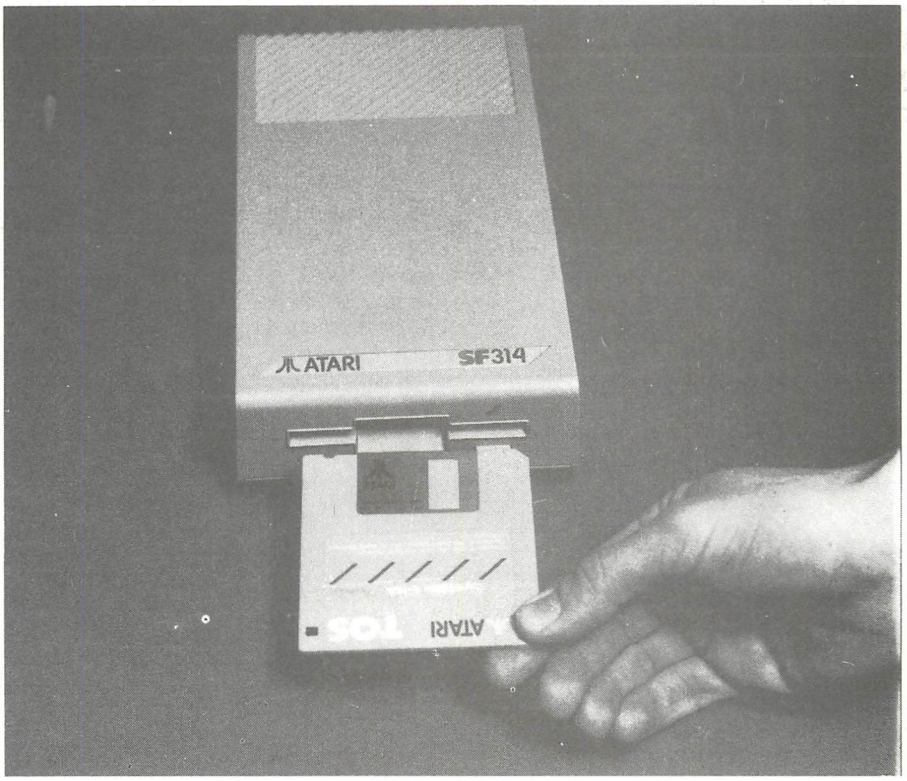
Mouse and joystick ports—1040ST

1.5 Preparations for start-up

For added convenience you may want to use a multiple-outlet strip to connect the various power cords used by the ST. Before plugging in the monitor and the ST power supply, make certain that all equipment is switched off. **Important: make sure you remove the inserts from your disk drive(s) before you turn on the power.**

The power switch is located on the back of the computer—number 4 in the photograph of the 520ST, and number 6 in the 1040ST photograph (see photographs on page 4).

The monitor's On/Volume switch must be set to "Off". The disk drive On/Off switch is in the drive's back. Insert the diskette labelled Language Disk into disk drive A (see photograph below):



Inserting a disk

Finally, plug in the power cords of the hardware. Then switch on the electrical components in the following sequence:

1. Monitor
2. Disk drive 2 (if you own one)
3. Disk drive 1 (520ST only)
4. Computer

Now you're ready to move on to Chapter 2.

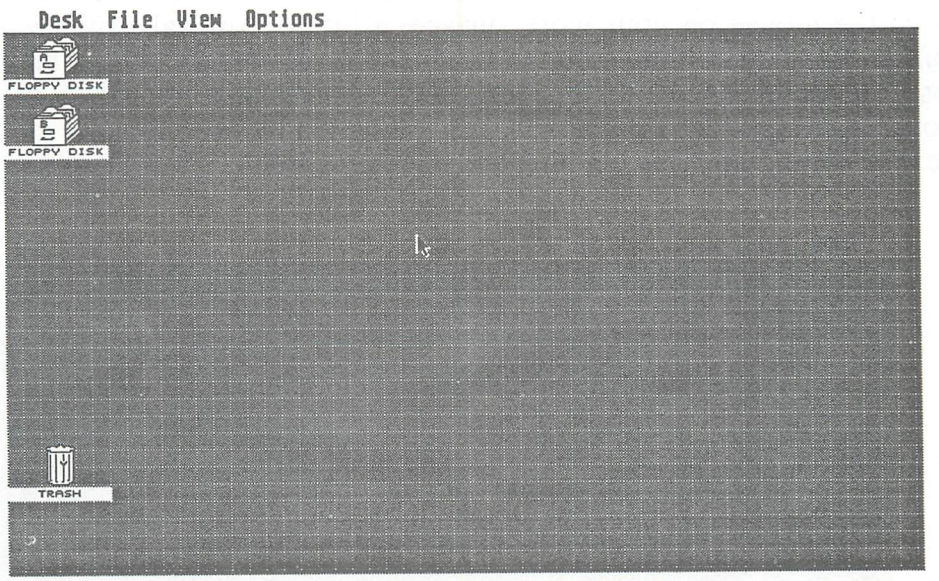
Chapter 2

Working with the ST

Working with the ST

2.1 Switching the power on

When you switch on the ST system's power, the following display pops up on the monitor screen. Your monitor screen may not look exactly as pictured. This depends on which monitor you have connected—color, monochrome or television set—as well as the *screen resolution* mode. If the objects appear twice as large as shown in the screen pictures, then you are in low resolution color mode. If the display does not appear, try adjusting the two upper knobs on the right side of the monitor to change the brightness and contrast. If you get no improvement, repeat the set-up instructions in Chapter 1.



The ST's *operating system* is activated when the computer's power is switched on. Put simply, the operating system tells the ST what it's supposed to do (how it is to operate). All messages and information which we receive from the ST are contained in this operating system. The ST is unable to function without an operating system.

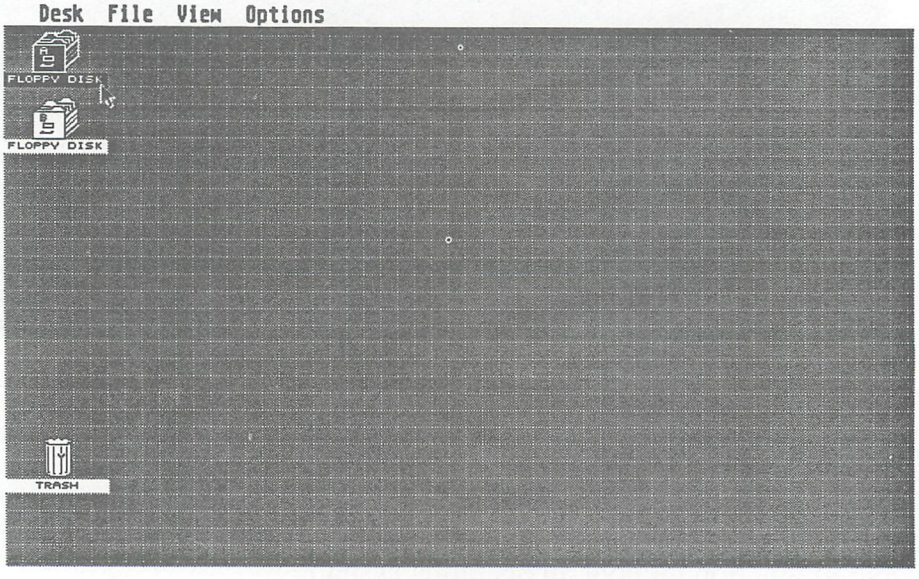
2.2 The GEM Desktop

A narrow white line appears on the top border of the screen and displays the words **Desk**, **File**, **View**, and **Options**. This is the *menu bar*.

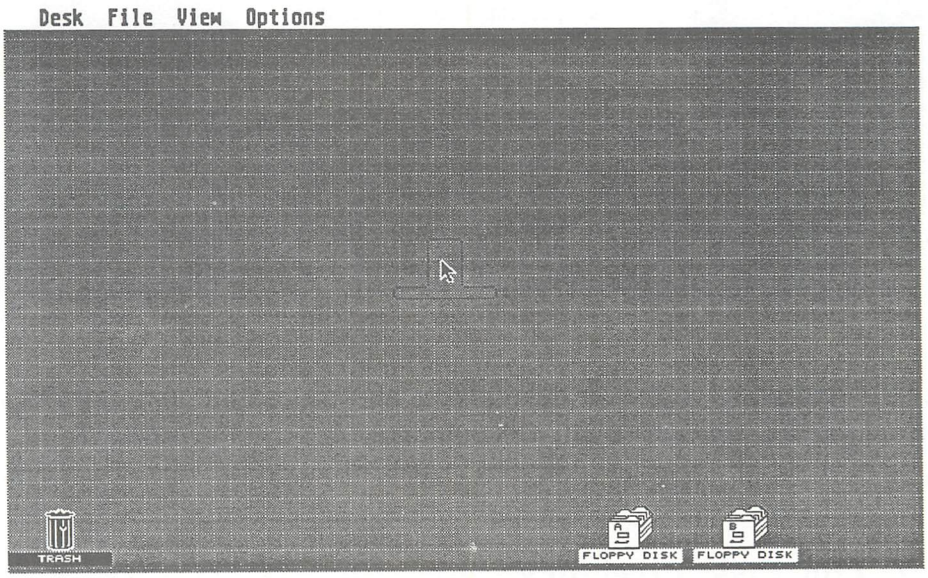
Beneath the menu bar are several pictures called *icons*. Imagine these icons as corresponding to items on your desk. Two disk drives are on the Desktop, and a trash can is on the floor. Think of the disk drives as file cabinets—note that the ST even displays your disk drives as file cabinet icons. We'll use this comparison later on in the chapter.

2.2.1 Screen arrangement

Don't like the arrangement of your Desktop? No problem. Notice that if you move the mouse on your work surface, a little arrow called the *mouse pointer* moves accordingly on the screen. Move the mouse pointer until it is positioned over one of the disk drives, then press and hold down the left mouse button. The ST acknowledges this action by reversing the color of the disk drive icon. This pointing and clicking action is know as *selecting*.



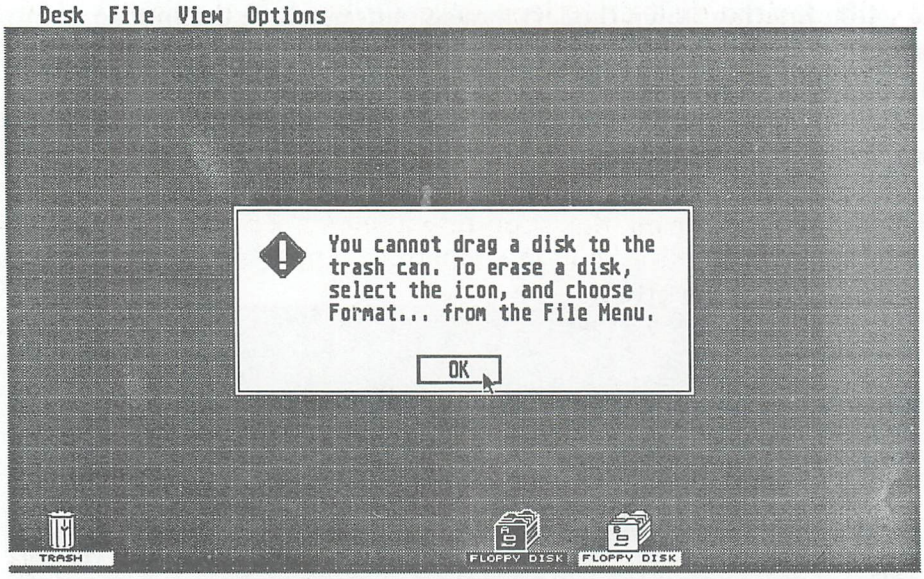
Now while holding down the left mouse button, move the mouse pointer to the right. A *ghost icon*, or silhouette, of the disk drive also moves to the right. If you release the mouse button now, the icon itself moves to the new location. At first only the ghost icon was moved, but as soon as you release the mouse button, the ST moved the icon to the new Desktop location. This procedure of moving an icon from one location to another is called *dragging* the icon.



Try moving the trash icon to another screen position. Dragging an icon is one of many tasks that the operating system performs. In reality, the dragging movement is not taking place on the monitor itself, but being processed and calculated inside your ST.

Be careful not to drag one of the disk drive icons over the other. If you do, the ST will attempt to perform a disk copy—and we aren't ready for that yet! If this happens accidentally, when the *alert box* appears on the screen, move the mouse pointer to the *dialog button* marked CANCEL and click this button. This will prevent an accidental disk copy.

If you try to drag a disk drive icon to the trash can, or the trash can to a disk drive icon, the ST will inform you that you cannot perform this operation. An alert box will appear on the screen to tell you that you made an error. To continue, move the mouse pointer to the OK dialog button inside the alert box and click the left mouse button.



The ST's operating system is designed to be very easy to use. It endeavors to be very polite and informative as you perform various operations.

2.2.2 The disk directory

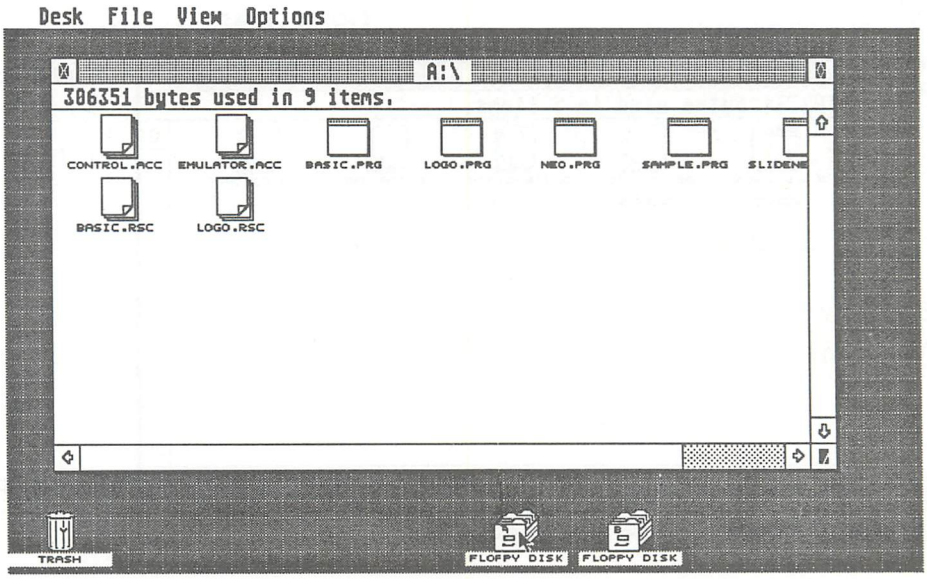
To find out which files are on a diskette, move the pointer to the desired disk drive icon (labeled `FLOPPY DISK`) and press the left mouse button.

The disk drive now appears in reverse video, indicating to the ST that you want to perform some operation with this device. Now press the left mouse button twice in quick succession. This is called *double-clicking*. The diskette in the disk drive should start to spin and the disk's *directory* will appear on the screen.

If the directory does not appear on the screen, don't worry. It takes practice to be able to double-click. Try to double-click the icon again. If you continue to have trouble, there is an alternate way to open a diskette.

Click the desired disk drive icon once so that it is displayed in reverse video. Move the mouse pointer to the word **File** on the menu bar. A *drop-down menu* will appear (see section 2.2.3 for details on and illustrations of drop-down menus). Now move the mouse pointer to the word Open and press the left mouse button again.

Selecting Open from the drop-down menu **File** is equivalent to double-clicking the disk icon. In either case, a new *window* will appear on the screen, as shown below:

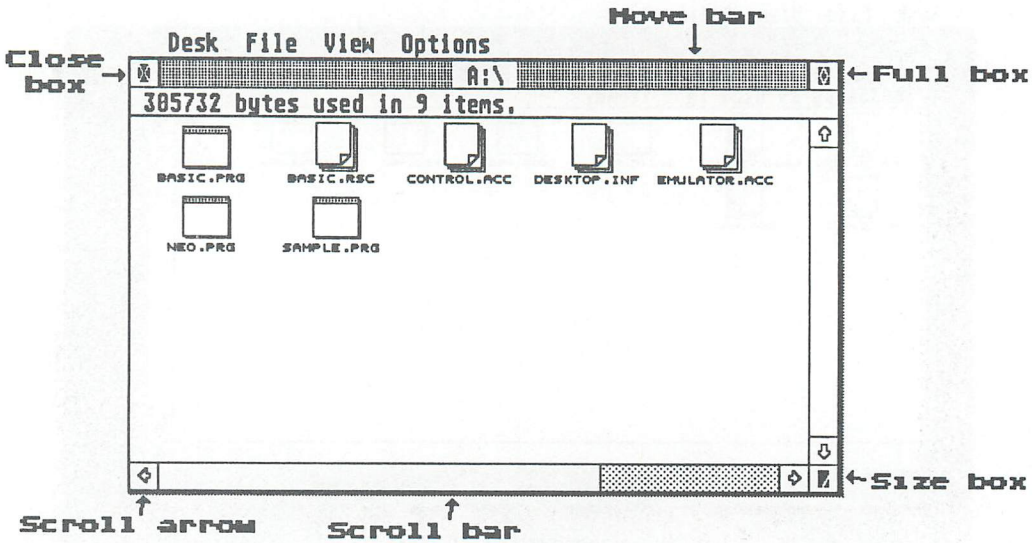


The disk currently in the disk drive contains individual program files or data files. A list of these filenames appears in the window. You may select the program with which you want to work from this group of files. This list is called the disk directory.

Now look closely at the disk directory window. Remember when we rearranged the icons on our Desktop? We can also change the window's location and size.

2.2.2.1 The window

A window is part of a screen which contains information about the work with which you are dealing. There are several functions which you can perform on an individual window. There is an illustration of a typical window at the bottom of this page, and the locations and names of these functions, or *attributes*. These terms and their functions shall be defined as you read on.

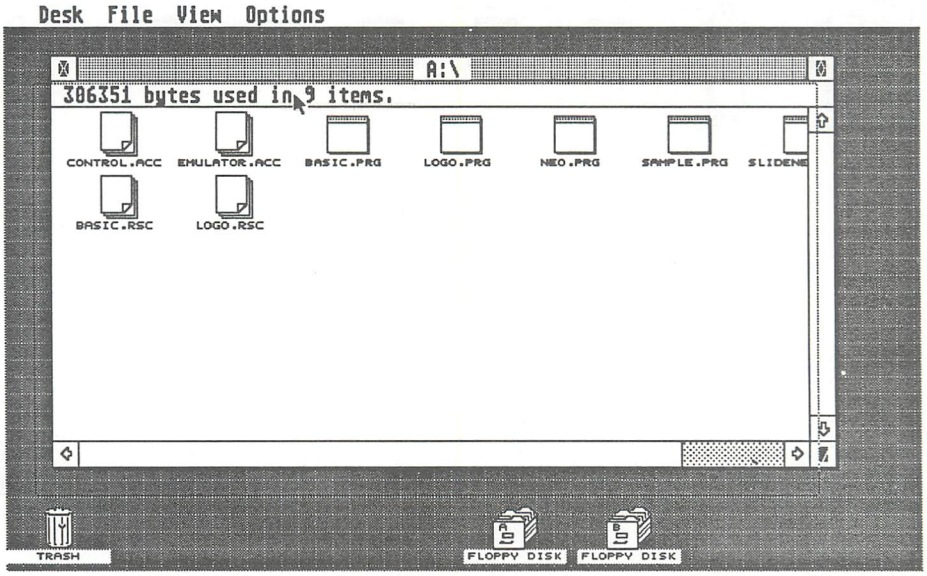


A typical window and its components

2.2.2.2 Moving the window

Move the mouse pointer to the top line, or *move bar*, of the window. This bar is labeled with the letter A:\ in the center. The A:\ means that this window is the directory of the diskette in drive A.

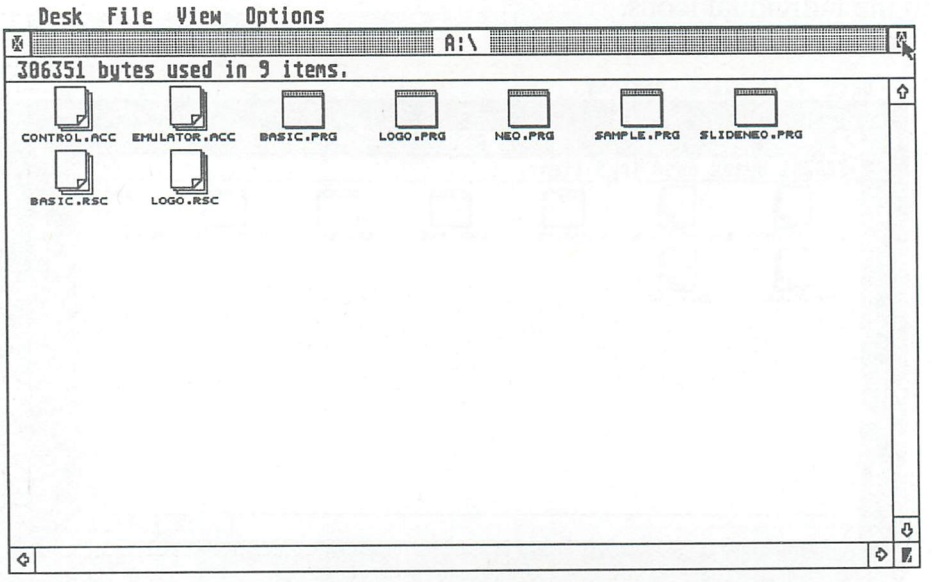
Position the mouse pointer on the move bar, press the left mouse button and hold it down. When you move the mouse, the entire disk directory window also moves. We can drag the entire window this way, just as we did earlier with the individual icons.



Now release the mouse button. The entire window is now shifted to its new position on the screen. You can even move it so that it "hides" the disk drive and trash can icons.

2.2.2.3 Changing the window size

You've probably noticed the other symbols along the border of your disk directory window. These symbols change the size of the window. To fill the entire screen, move the mouse pointer to the box at the upper right corner of the window (the *full box*). Then click the left mouse button (do not hold it down this time, just click it once). Now the window occupies the entire screen.



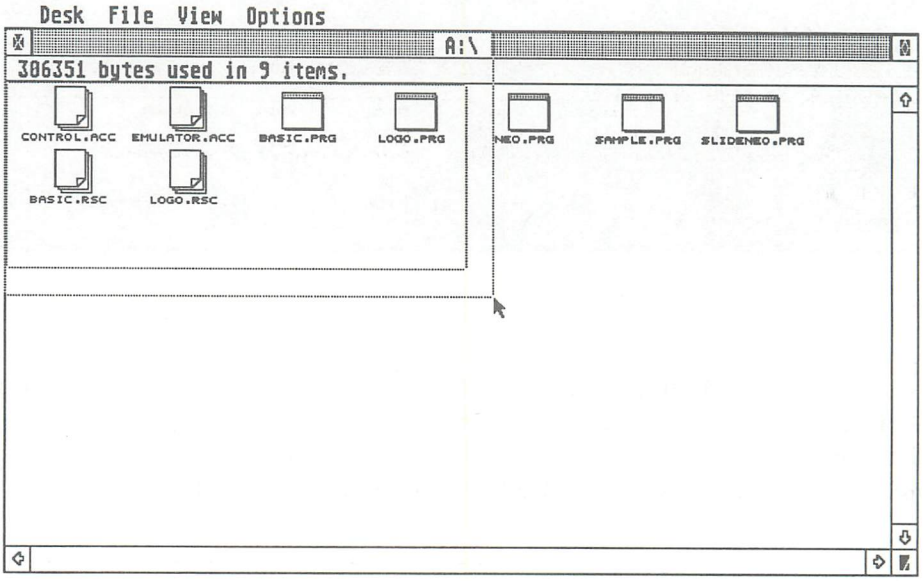
If you click the full box again, the process is reversed. Now we see only part of the disk directory.

Note: The full box is similar to an On/Off switch. When it's in the On position the window is enlarged. Clicking it Off again returns the window to its previous size.

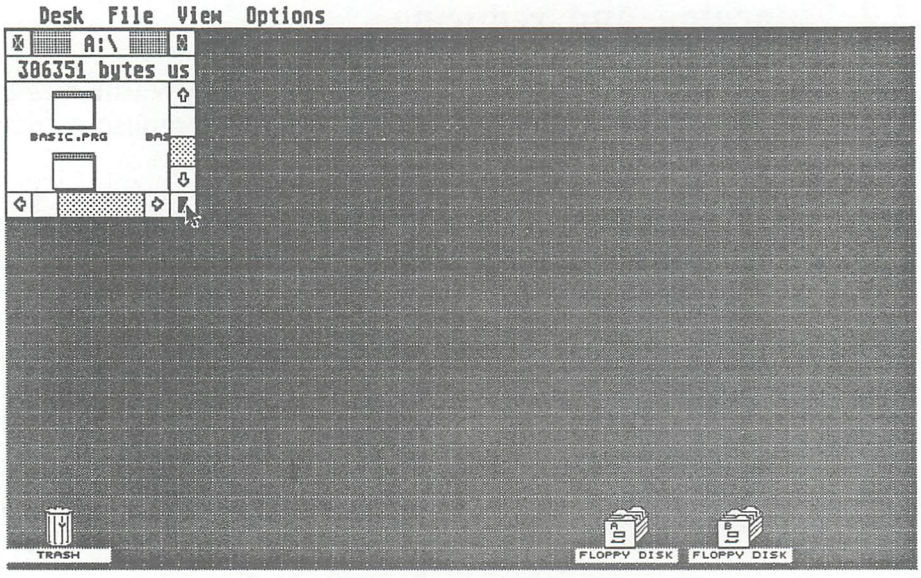
2.2.2.4 Enlarging and reducing

The small symbol at the bottom right corner of the window (the *size box*) gives you more control over the window's size. Move the mouse pointer to this box, press the left mouse key and hold it.

Now move the mouse back and forth on the desk. You can change the size of the disk directory window quickly and easily in this way.



Reduce the window to its smallest size using the size box. Hold down the left mouse button while the mouse pointer is on the size box, move the mouse pointer to the upper left-hand corner of the screen, and release the mouse button.

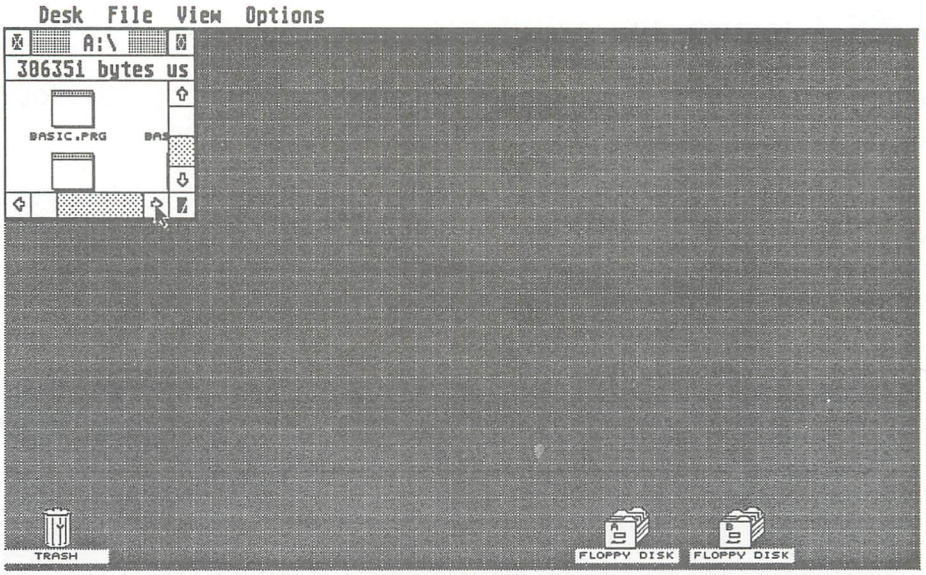


Now we only see a small portion of the directory. If you want to enlarge the disk directory, drag the size box to the right with the pointer. You also may return the disk directory to its original size by the repeatedly clicking the mouse on the size box, or by dragging the size box down and right.

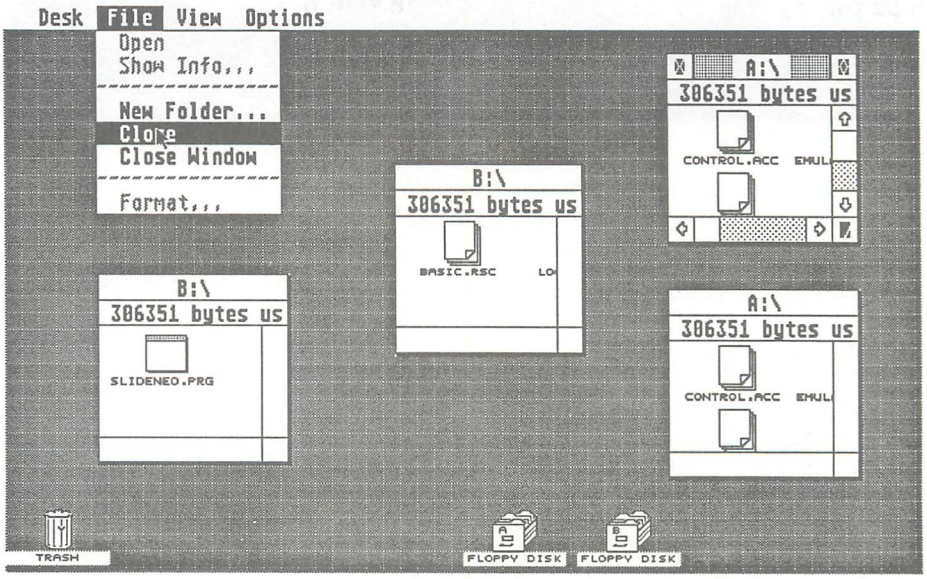
2.2.2.5 Scrolling

You don't really need to do all this stretching. The *scroll arrows* , two on the right of the window and two on the bottom, let you scroll through the window's contents. A portion of the bars running along the bottom and right edges of the window, or *scroll bars*, will be shaded when files are hidden from view. The unshaded portion is what is being viewed presently, the shaded portion is the part that is now visible at the present time.

Make the disk directory window as small as you can. Move the mouse pointer to the scroll arrow at the bottom right, and press the mouse button. The next segment of our disk directory becomes visible. Do the same with the other scroll arrow. Although the window can only show a small picture right now, you can see the entire directory by scrolling down and right with the scroll arrows and scroll boxes.



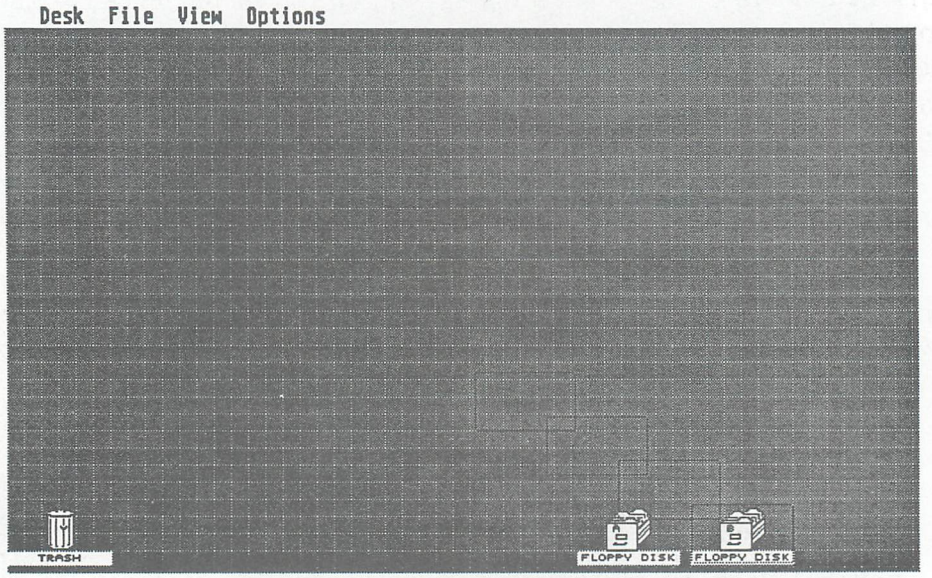
"Great," you wonder, "but what good is that?" Well, this allows you to have several disk directories on the screen at once. The *scrolling* allows you to view and compare the contents of the different disk directories.



2.2.2.6 Closing the windows

The upper left corner of our window is called the *close box*. The close box removes the window from the screen, i.e., closes the window.

Click the close box with the mouse. The disk directory vanishes and the desk is clear. Now, as when you turned the power on, there are only two disk drive icons and one trash icon on the desk. The window with the disk directory has "disappeared" into disk drive A.

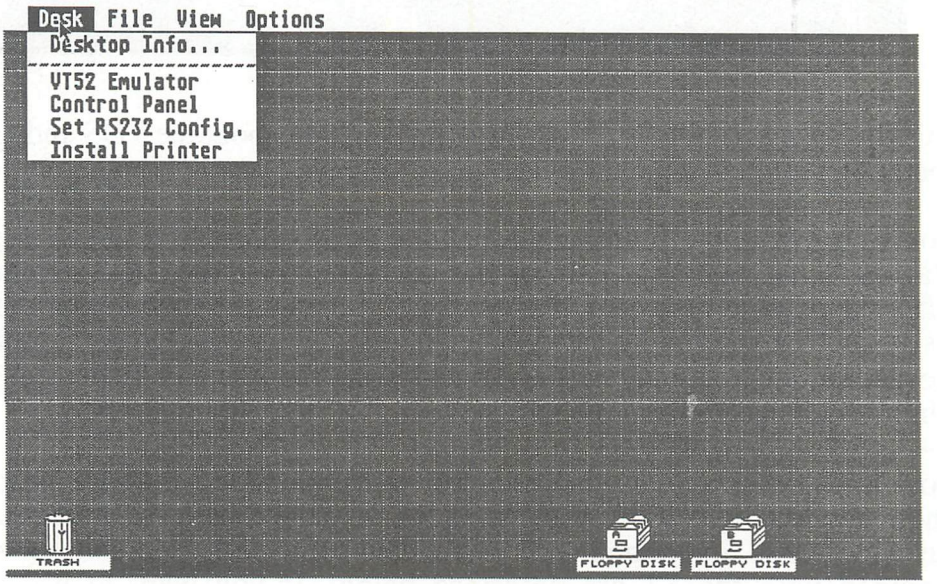


2.2.3 The drop-down menus

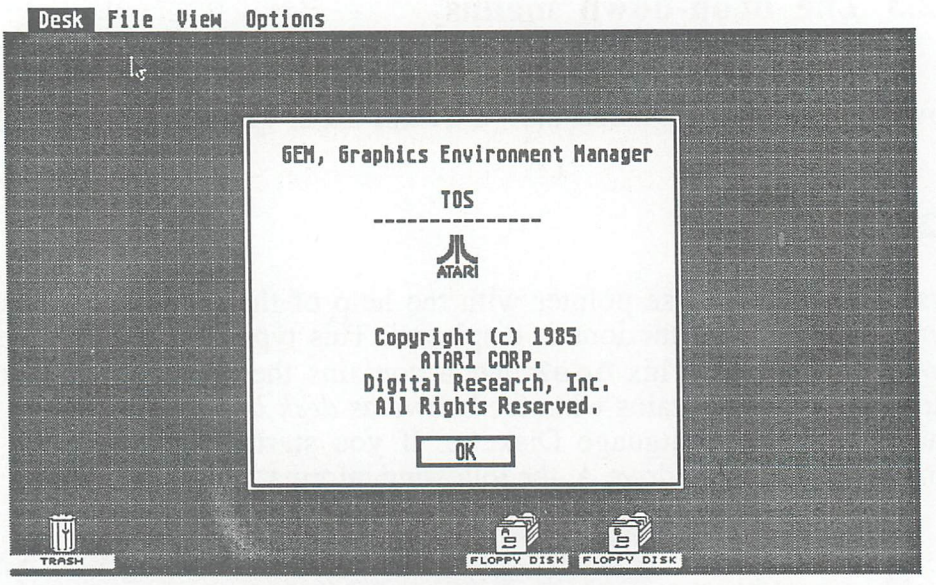
Now let's look at the menu bar—**Desk**, **File**, **View**, **Options**.

2.2.3.1 Desk

First, move the mouse pointer with the help of the mouse to **Desk**. A *menu* of available functions is displayed. This type of menu is called a *drop-down menu*. This **Desk** menu contains the Desktop Info... selection. It also contains what are known as *desk accessories*, which are loaded from the Language Diskette. If you started your ST with the Language Diskette in drive A, the following information can be seen:



Move the mouse pointer to **Desk** on the menu bar, then move the mouse pointer to Desktop Info.... The Desktop Info... will be displayed in reverse video when the mouse pointer moves over it. Select Desktop Info... with the mouse and the following appears:



This displays a *dialog box* containing information about the TOS and GEM operating systems in the ST.

You've probably already heard about GEM (Graphics Environment Manager). GEM is the part of the operating system that controls the different graphic functions (for example, it draws the four corners of the disk directory window). Once you have read the information on the screen, move the mouse pointer to the OK and press the left mouse button. This window disappears from the screen.

Now we'll describe the other choices we have under the **Desk** heading. These are desk accessories that are loaded from the diskette in drive A when the computer is first turned on. The standard set of desk accessories that are included with the ST are described below. If you want to use these functions of your ST later, you'll find the necessary information here.

VT52 Emulator

If you select the VT52 Emulator with the mouse, the graphic screen disappears and the following message appears:

```
**********************************
| Atari VT52 Terminal Emulator  |
| (c) Atari Corp.               |
**********************************
Press:
1) UNDO to return to desktop.
2) HELP to configure terminal.
```

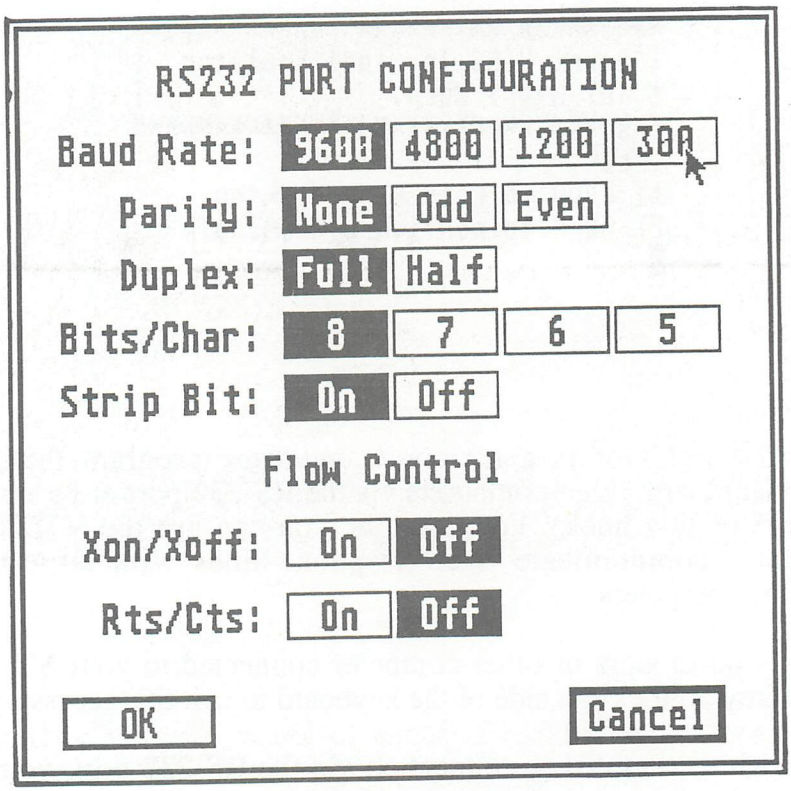
The VT52 emulator is a *terminal emulator* program that lets you communicate with other computers via the RS-232 port or by modem (see Chapter 5 of this book). For example, you can use the VT52 emulator program to communicate over telephone lines with large corporate mainframe computers.

If there is no modem or other computer connected to your ST, press the <Undo> key on the right side of the keyboard to exit this accessory.

If you wish to establish a connection via the RS-232 port, you not only have to click on VT52 Emulator with the mouse, but also configure the Set RS232 Config. for proper transmission and reception. You can do this directly from the running VT52 emulator program if you press the <Help> key on the right side of the keyboard.

Set RS232 Config.

You can also select Set RS232 Config. in the **Desk** drop-down menu. This option displays another menu on the screen:

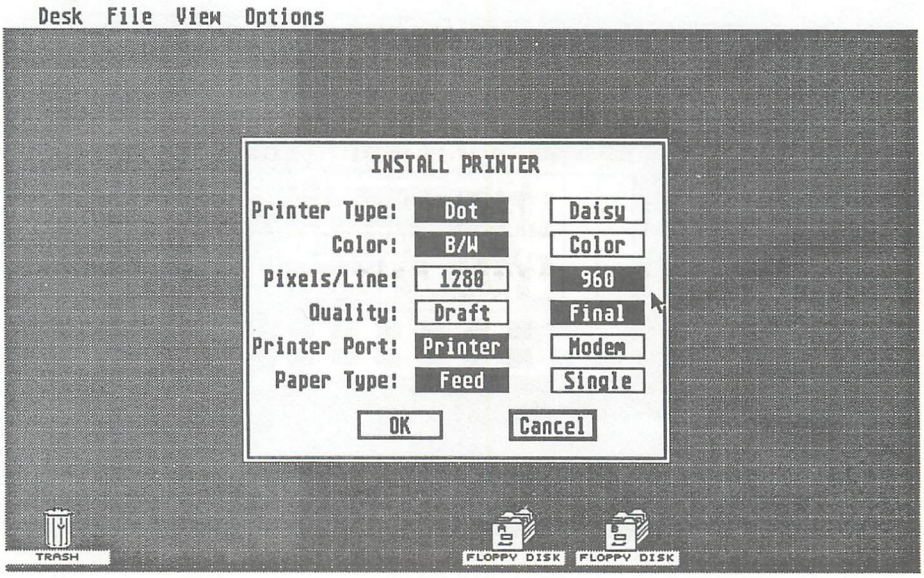


Now you may select different options located next to one another (for example, transfer rate at 9600, 4800, 1200 or 300 baud). Move the mouse pointer to the desired option and press the left mouse button. The different options are used to configure the RS-232 port on the back of the computer. This port is used to connect various peripherals to your ST, such as printers, modems and plotters.

The selected box is highlighted in reverse video (e.g. white type on black background) to confirm the choice. Once the configuration is complete, move the mouse pointer to the OK dialog button at the bottom left and click to confirm. If the configuration was wrong, click the CANCEL dialog button on the right hand side.

Install Printer

This option lets you specify how your printer is set up. If you want to set up your printer at this stage, then follow these directions:

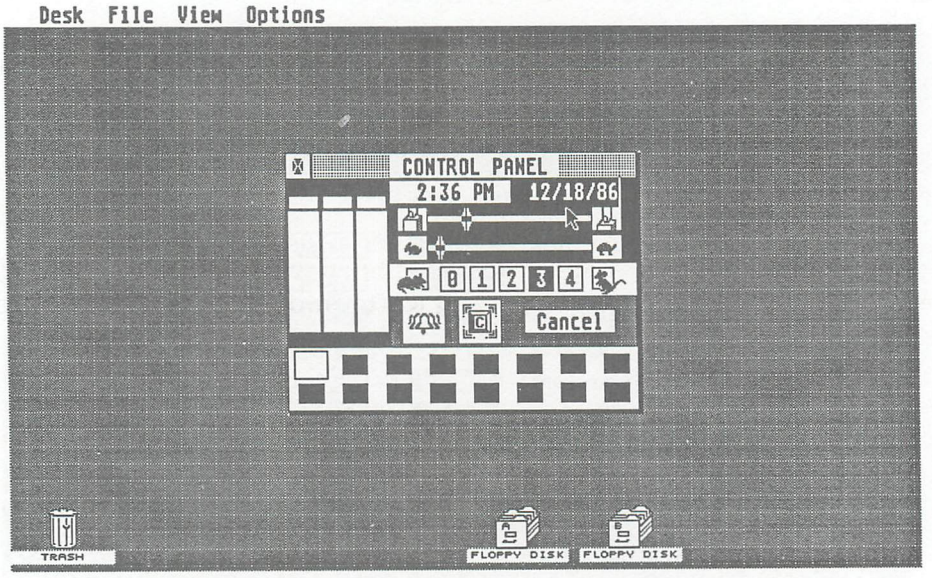


Here `Install Printer` needs to know whether your printer is dot-matrix or daisywheel, color or black & white. It's important to set `Pixels/Line` to `960`, and `Quality` to `Final` when using Epson and Epson-compatible printers for hardcopy (screen dumps). Otherwise the ST prints only part of the screen. If your printer is designed to use the RS-232 interface, set the `Printer Port` to `Modem`. Centronics interfaces (parallel) use the `Printer` selection. Also select whether you are using continuous form paper (`Feed`) or single sheets of paper (`Single`).

If you're a little confused by all of this "computerese", don't worry. We'll explain the operation of printers in detail later on.

Control Panel

The Control Panel is another function of the **Desk** menu:



You can use the Control Panel to set:

- time and date
- key repeat delay
- keyboard repeat speed
- key click
- bell
- color
- mouse double-click speed

If you have a color monitor, color tint adjustments are made with the three RGB (red-green-blue) *color palette controls* on the left side of the Control Panel.

Monochrome monitor users have only two screen colors available: white on black, or black on white, selected with the three *palette controls*. Move the controls with the mouse pointer in the direction opposite the shade you desire.

Here are the rest of the Control Panel options:

Clock : Move the mouse pointer into the clock window and click the left mouse button. The time is displayed in reverse video to mark your selection. Erase the old time with the <Backspace> key (or press <Esc>), and enter the current time. You confirm this by clicking the time field with the left mouse key, or by pressing the <Return> key.

Calendar : Same as the clock function. Your ST will display any data you enter, so be sure to enter it correctly.

Keyboard Response : Moving the slider on the upper side adjusts the delay between keypress and key repeat. The lower slider controls the speed at which the keys repeat.

Mouse Click Response : The icons are self-explanatory. Position 0 (mouse at rest) produces slow double-clicking. Position 4 requires quick reflexes to do a double-click. Position 2 or 3 should be right for you.

Audio Feedback: These two symbols may be switched on or off by the mouse. The bell rings to indicate an error; the key click sounds when you depress a key. Although both functions are valuable as controls, we advise you to turn the volume adjustment on the monitor as far down as possible, instead of adjusting volume from the Control Panel.

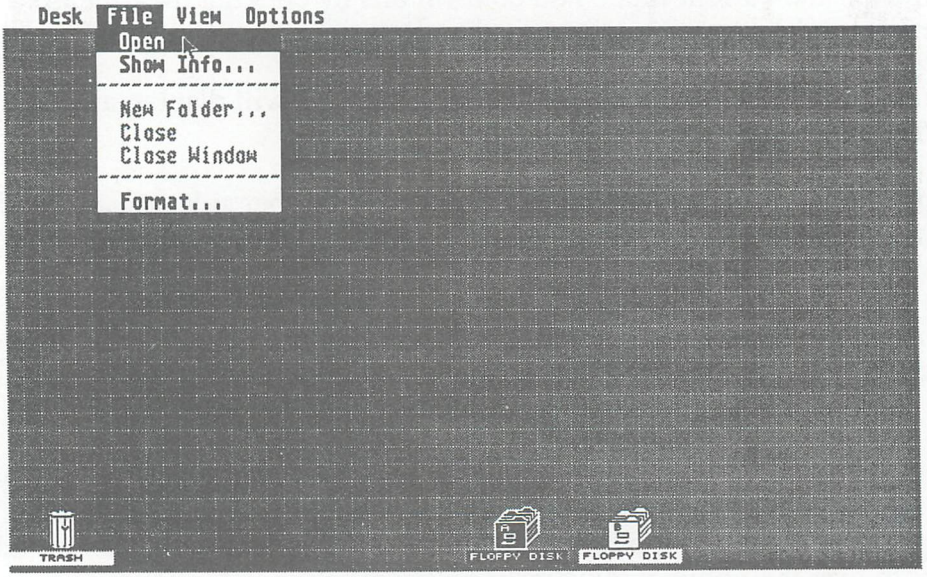
We mentioned the color palette (RGB) sliders earlier. Using them, you can store up to 16 colors in the lower part of the Control Panel, and call them up again with the mouse. (These 16 blocks have no meaning on monochrome monitors).

Finally, we should mention the CANCEL dialog button in the control panel. If you don't like your changes, click the CANCEL dialog button. All values except date and time are immediately changed to their original status.

You may move the Control Panel window around the screen by clicking on the hatched area of the window and pulling. Close the Control Panel window the same way you closed the disk directory window—click once in the close box at the upper left corner.

2.2.3.2 File

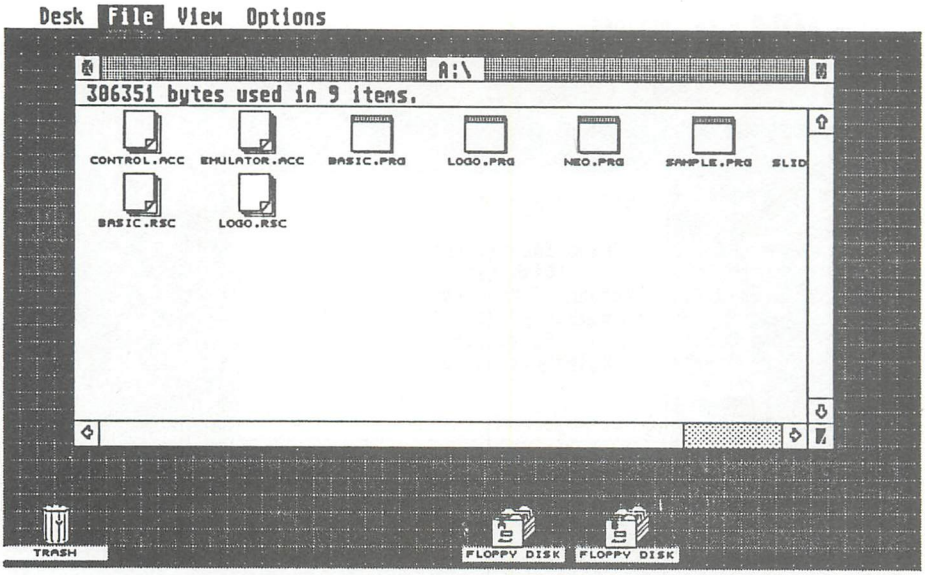
Now we'll look at the **File** menu. Click drive A's icon with the mouse pointer to select drive A, then move the pointer to **File** on the menu bar:



Open

The Open option allows us to run a program or examine the contents of a diskette, folder or file. This function is useful because some users, especially beginners, have difficulty when double-clicking the mouse button, even at low speed. Open lets you open a selected disk or file from the **File** drop-down menu with only a single click.

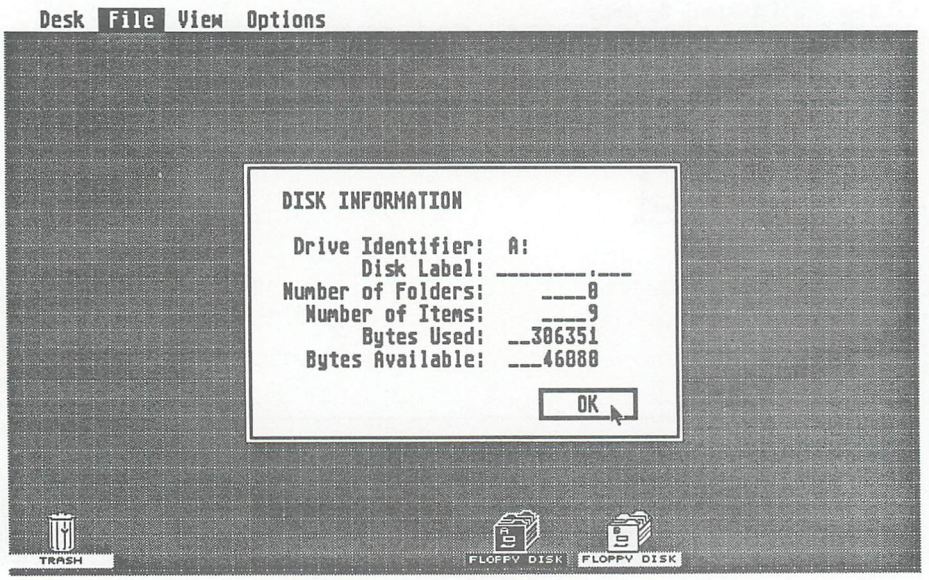
If you point to Open with the mouse pointer and then press the left mouse button, the directory of disk drive A is displayed:



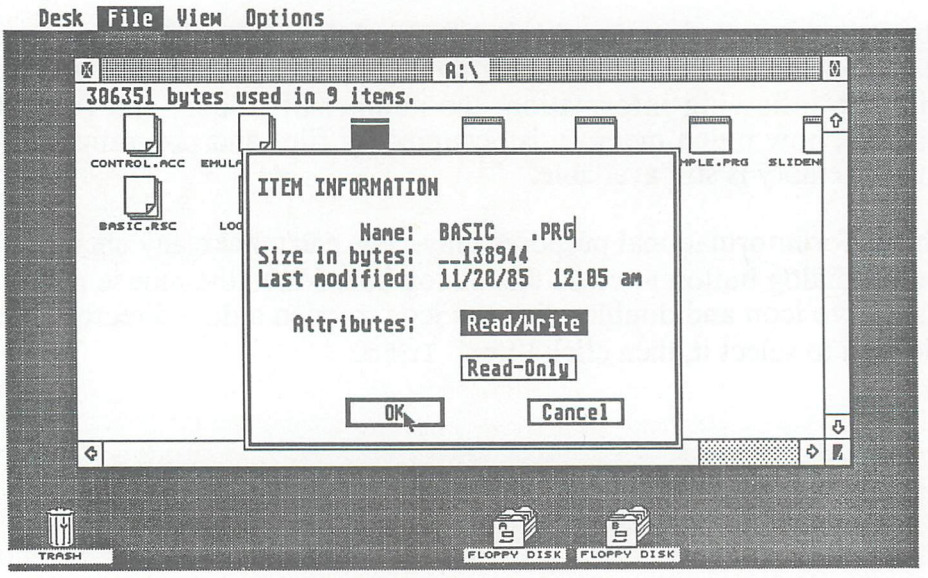
Show Info

The Show Info function displays information about a Desktop icon or a file. If you select a disk drive icon, then Show Info, a dialog box appears with the following information: the number of folders and files on the diskette, how much memory is occupied by files and programs, and how much memory is still available.

This is for informational purposes only. You can't make any changes. Click the OK dialog button to close this dialog box. Move the mouse pointer to a disk drive icon and double-click the icon to open a disk directory. Click a file icon to select it, then click Show Info.



Disk information

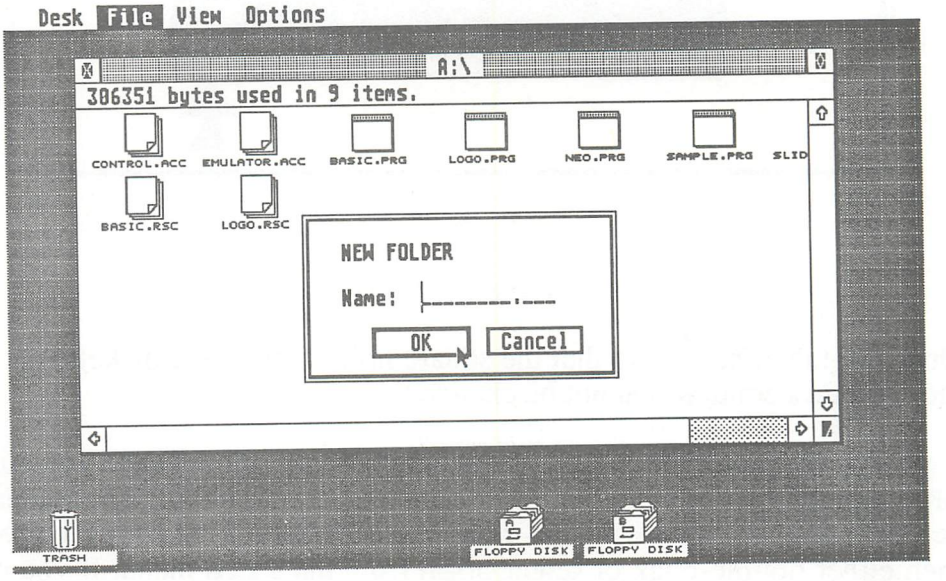


Item information

To change a filename, press the <Backspace> or <Esc> key to delete the current filename. Then use the keyboard to enter the new filename. You may also designate files as READ/WRITE or READ ONLY. Confirm by clicking the OK dialog button.

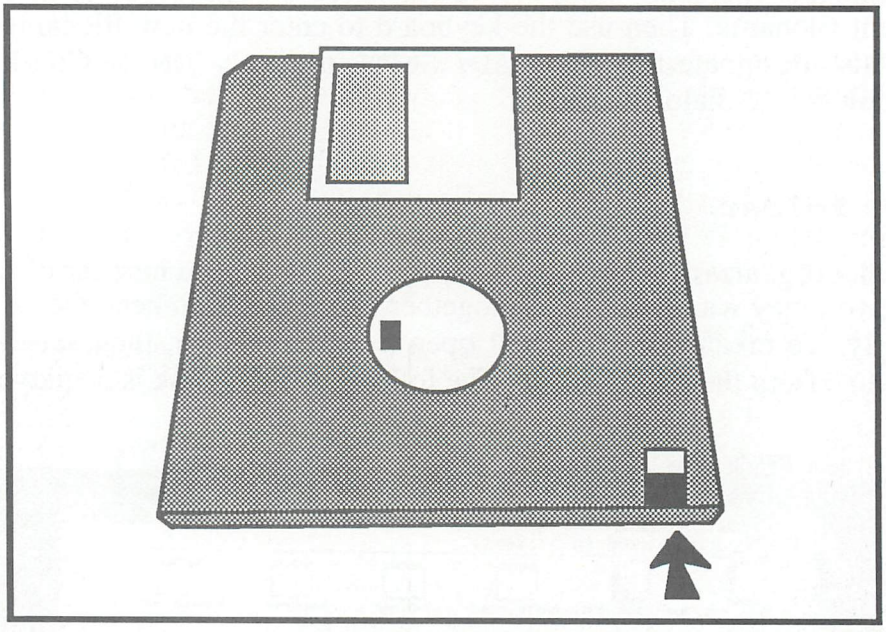
New Folder

A folder organizes your files on a diskette. If you have a number of related files, you may want to put them together in one folder to keep the Desktop orderly. To create a folder, first open a disk directory, then select New Folder from the **File** menu. The following dialog box is displayed:



Enter a name for the folder, such as ACCESSORIES. Then click the OK dialog button. The new folder will be created on the currently open disk drive. After it is created you will see an icon representing the folder on the disk directory.

If you're still using the original Language Diskette, it is *write-protected* to keep changes from being made accidentally. After you clicked the OK dialog button, the ST informs you with an alert box that it cannot perform this function because the disk is write-protected. The *write-protect tab* is a small plastic slider on the back of the diskette:



Write-protect tab

Once the tab is moved so that the square hole through the diskette is open, the diskette's contents cannot be changed.

If your write-protect was "up" (the square hole was closed) and you clicked OK, you now have a folder with the name ACCESSORIES. The folder can be opened in the same way that a diskette is opened. First select the folder, then either double-click or select Open from the **File** menu. If you open the ACCESSORIES folder you will see that it is empty. We'll see how to copy files into this folder in Section 2.4.

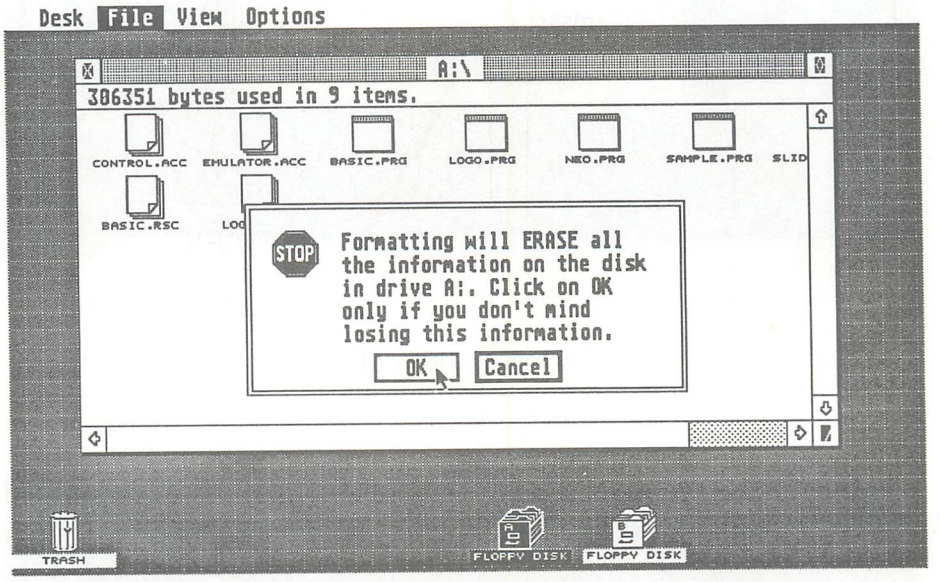
Close and Close Window

These two options are methods of closing a window. Simply click either of these options from the **File** menu. Clicking the window's close box is another way of closing the window.

It's a good idea to keep windows closed when they are not in use, since they can clutter up the Desktop.

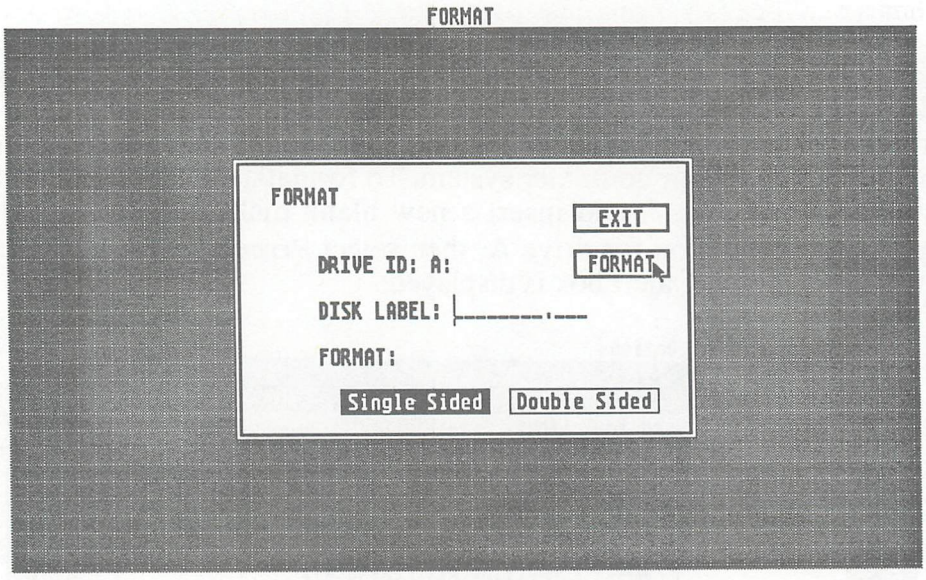
Format...

Formatting does more than erase diskettes. When you purchase blank diskettes, they are really blank—they cannot be used "as is" in the ST. Formatting divides or formats the diskette into tracks and sectors for data storage on a particular computer system. To format a diskette, first remove the diskette in drive A and insert a new blank diskette. Next select the floppy disk drive icon for drive A, then select Format... from the **File** menu. The following alert box is displayed:

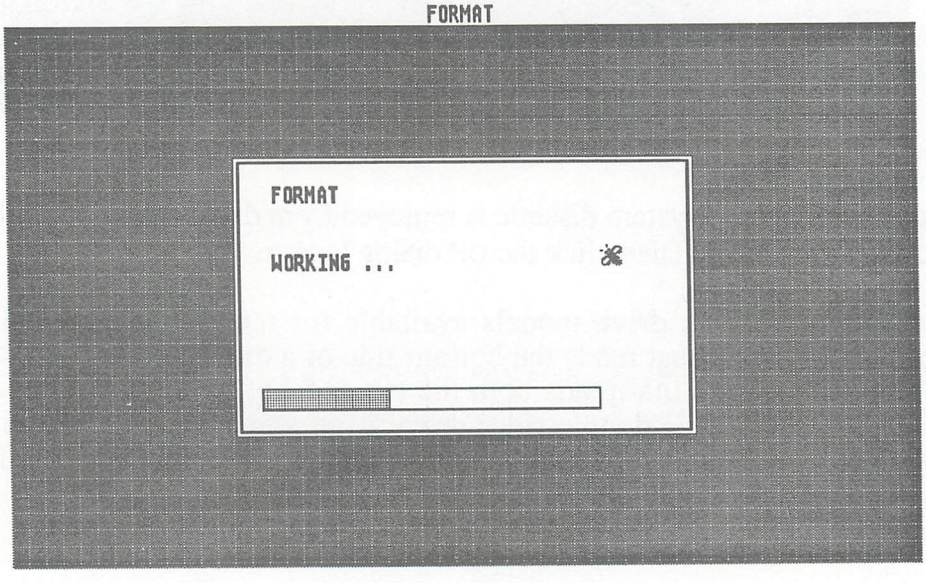


Make sure that the system diskette is removed from drive A and a new blank diskette is inserted. Then click the OK dialog button.

There are two disk drive models available for the ST. The SF354 is a single-sided drive that reads the bottom side of a diskette. The SF314 is a double-sided drive that reads both the top and bottom of a diskette. The 1040ST has a built-in double-sided drive. After you click OK the following dialog box appears. Insert a diskette, input a disk name and select single- or double-sided, and click FORMAT:



Format program

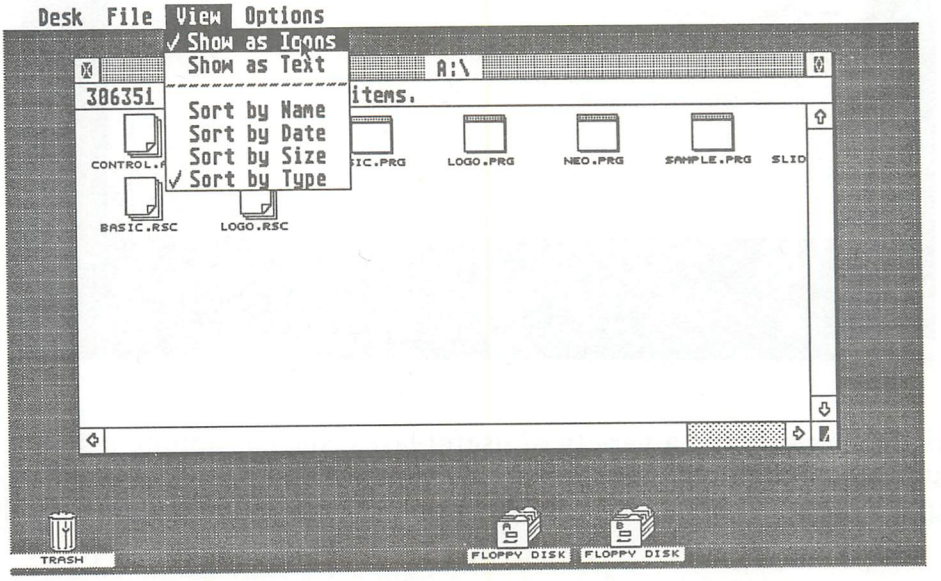


Formatting in progress

When the process is complete, a message appears: e.g. `357376 bytes are available` (for a single-sided diskette). You can now format more diskettes or `EXIT` this procedure.

2.2.3.3 View

Let's open a diskette containing files and have a look at the menu marked **View**:



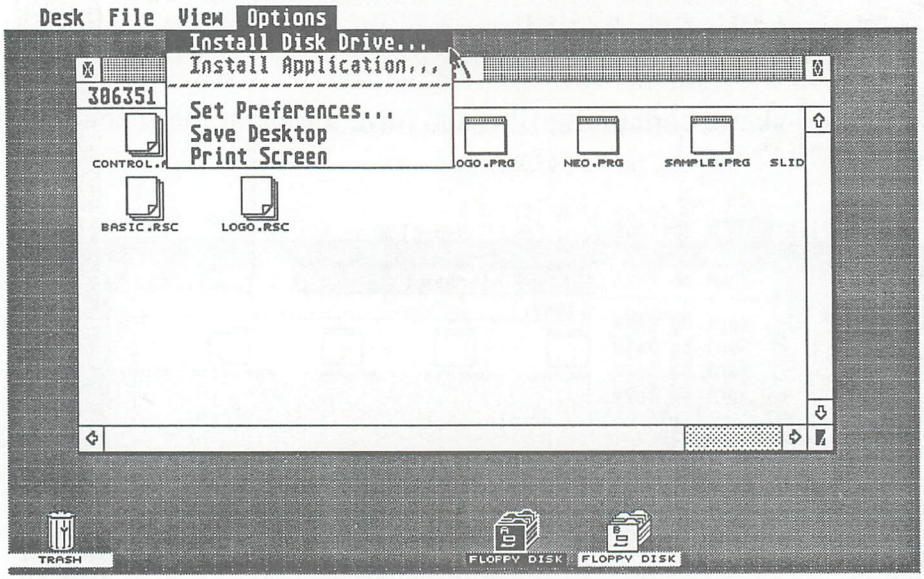
Show as Icons
Show as Text
Sort...

Here you choose how the diskette directory appears on the screen. Also, the directory can be sorted by filenames, dates, file sizes or file types.

Your choice is executed immediately, so the change may be seen as soon as you select your choice. The system marks your choice with a check mark. Try out the various options to see the results.

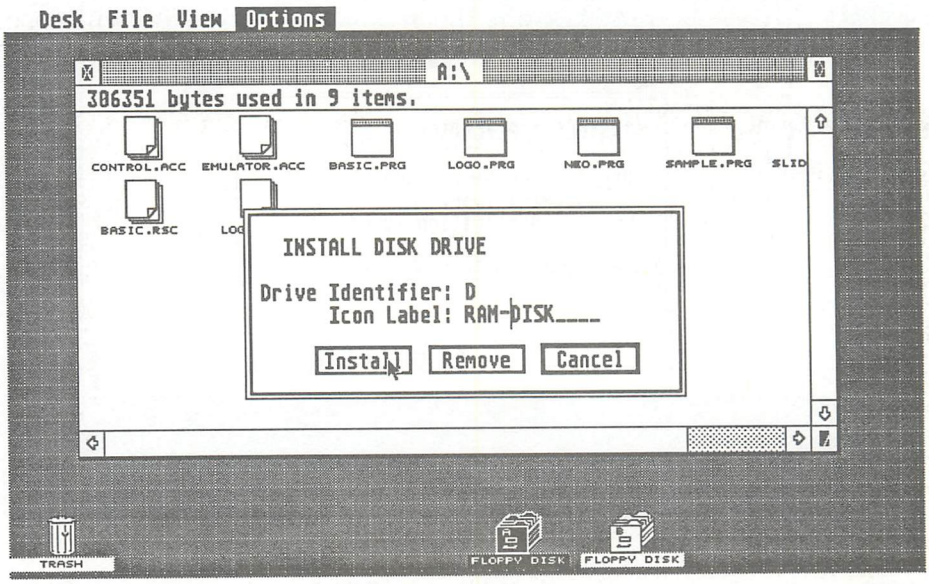
2.2.3.4 Options

Select disk drive A before choosing this menu. The following will appear:



This menu performs a variety of useful tasks, such as adding a disk drive, installing an application program, saving your control panel and Desktop settings onto a diskette, and getting a printout of the current screen display.

Install Disk Drive...

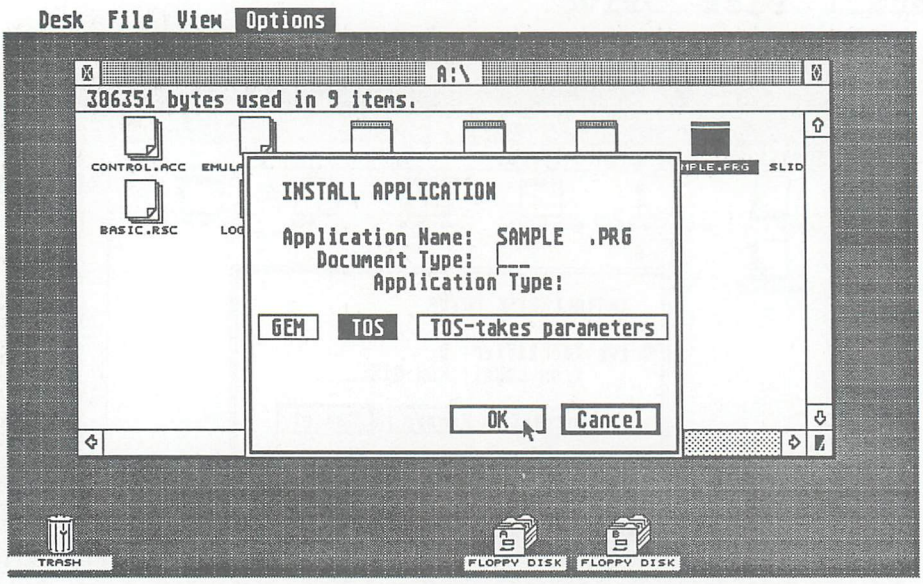


You can now change the disk drive name under a drive icon, and even the diskette name itself. This is useful for adding hard disks and RAM disks (more on these later). You can also Remove the diskette from the screen. Be careful with this selection—once you have erased the two drives, you cannot use your system diskette anymore. Try renaming the icon label for FLOPPY DISK to 3 1/2" DISK.

Install Application...

This function appears in grey on the screen, and cannot be accessed at the moment. Only functions with black lettering can be clicked.

To use this function, you have to select a program from the directory (programs have .TOS or .PRG *extensions* which follow the filename itself, and are marked as block icons).

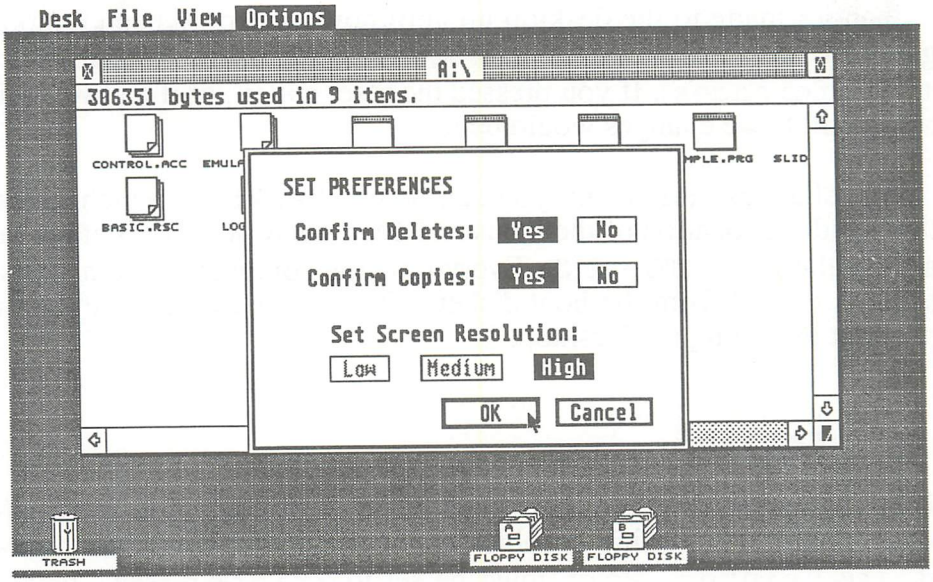


The ST can run both programs which use GEM, and programs which don't use this graphic interface. These programs run without icon control from GEM, which means that they run under the TOS operating system only, bypassing GEM altogether.

`Install Application...` allows you to determine what type of file will directly start programs (those with `.PRG` extensions), and allows you to choose the operating mode (`GEM`, `TOS` or `TOS takes parameters`).

We can now enter the document type. This tells the operating system that you want files with a particular filename extension to start the application program you've installed when these files are opened. Programs (`.PRG`, `.TOS` and `.TTP`) can be automatically started by opening them, but data files can also be started with a "document type" extension. When these files are double-clicked, the installed application program, then the files themselves, are loaded in to the computer. For example, give a program that runs in ST BASIC a `.BAS` extension. Select `Install Application...` and then install `BASIC.PRG` with a document type of `BAS`. Now when you select a file with a `.BAS` extension, the ST will load `BASIC.PRG`, then the file you selected will load and run.

Set Preferences...



The ST always asks if you actually want to delete a file, or if you wish to copy a file or diskette. If you are absolutely certain of how to handle deleting or copying of files, you may click the NO dialog buttons at `Confirm Deletes` or `Confirm Copies`. This will speed up these procedures— **but remember, you will run a much greater risk of losing your valuable data.**

As mentioned before, the ST has three different *screen resolutions*. You can only use `High` resolution on a monochrome monitor. You can only use `Low` and `Medium` resolutions on color monitors.

Entries are confirmed by clicking `OK`, and cancelled by clicking `Cancel`.

Save Desktop

The changes made to the desktop up until now have only been temporary (e.g., Control Panel, RS-232 Config., Show.., SORT..., Set Preferences). If you pressed the reset button, or if a power failure occurred, all these changes would be lost.

To make changes permanent, you can Save Desktop, which saves all options and the location of the icons and open windows. This data is saved to a file called DESKTOP . INF. Every time the computer system starts up, this file is loaded from the boot diskette. Thus your changes can be made permanent by saving the Desktop.

Print Screen

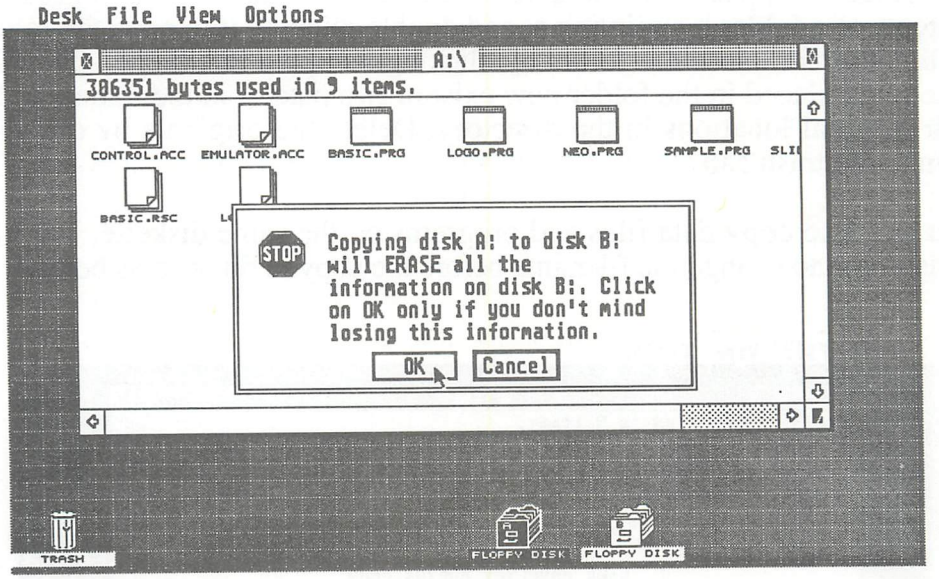
This option prints the current screen on your printer. You must have a printer capable of printing graphics connected to the Centronics parallel port. The Printer Port must be set to Printer (see Install Printer from the **Desk** menu). This screen printout is known as a *screen dump*.

Pressing <Alternate> and <Help> simultaneously on the keyboard will also produce a screen dump.

2.3 Copying diskettes

Now let's copy the Language Diskette. If you haven't formatted a blank diskette yet, turn back to page 35 (Format). **Note**: Even if you have a double-sided drive, you must still format the backup as a single-sided diskette. The double-sided drive will read both formats. The one supplied by Atari is formatted as single-sided so that both drive models can read it.

Click the mouse pointer on drive A, and drag drive icon A to disk drive B. A confirmation message appears on the screen before the copy procedure begins. Click OK to continue, or CANCEL to exit.



If you click OK, another dialog box will appear. The copying procedure from here is simple if you have two disk drives. Insert the original diskette (in this case the Language Diskette) into drive A, and the destination diskette (the formatted diskette) into drive B. Then click COPY.

If you have only one disk drive, click COPY, then exchange diskettes in drive A when the ST tells you to switch diskettes (the original diskette is A, the destination diskette is B). The ST loads the contents of the original diskette into memory, then saves these contents to the destination diskette.

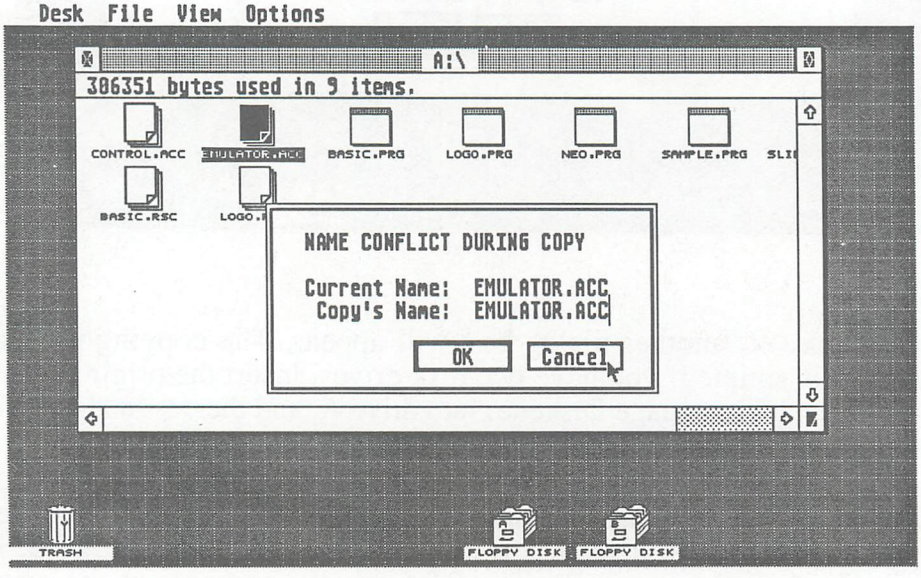
2.4 Copying and deleting data files and programs

Individual files are copied by clicking an individual icon in the directory, and dragging it either to the drive B icon or the drive B directory. A message similar to the copy message appears, and you perform the copying as outlined in Section 2.3.

You can also copy programs and files into folders. Drag the files by holding down the mouse key and moving the ghost icon onto the folder.

Each time you want to place a file into the folder, a message requesting confirmation appears on the screen. Click OK. You can then check the contents of a folder by pointing at and double-clicking it to open the folder. If all is correct, close the folder by clicking its upper left corner. The files which you placed in the folder now exist in two places: in the folder, and in their original locations in the directory. Delete the originals by dragging them to the trash can.

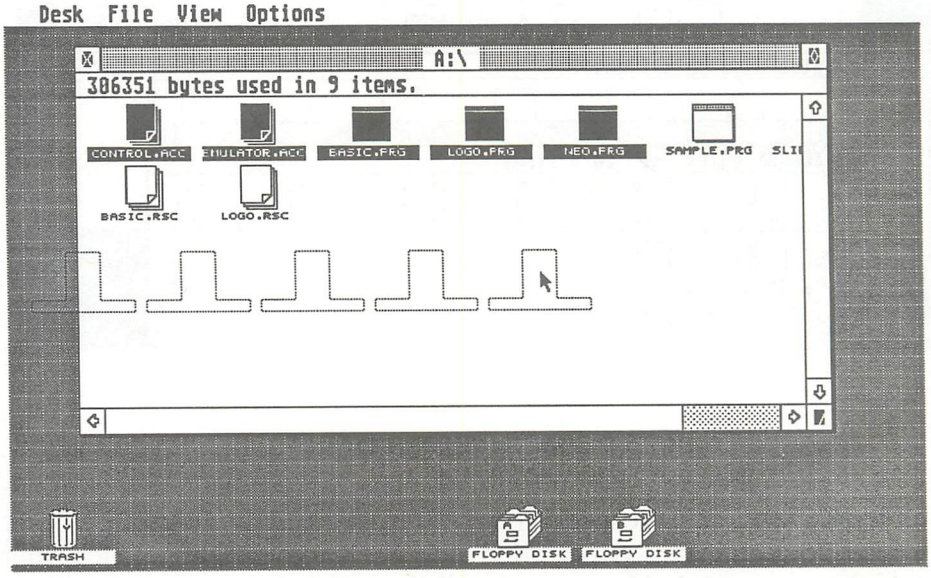
You can also copy data files and programs on the same diskette. Your ST advises you to change the filename before the copy is finished as below:



Files can be deleted individually by dragging them to the trash can. Open the disk directory and drag the program or file to be deleted to the trash can (place the file over the trash can icon).

There is a danger in deleting files. If you accidentally throw away some papers at home, you can search the wastebasket and (hopefully) find the discarded item. But once you drag a file to the ST's trash icon, it's irretrievable–gone forever. **Be careful!**

Several files can be deleted at once. Mark the files by framing them, i.e., pressing and holding the mouse key starting from top left of the window. This "lassos" the group of files—they can be copied or trashed all at once. This group of ghost icons will move together for copying or deletion. Use the mouse arrow to position the group over the correct destination icon (folder, diskette or trash can).

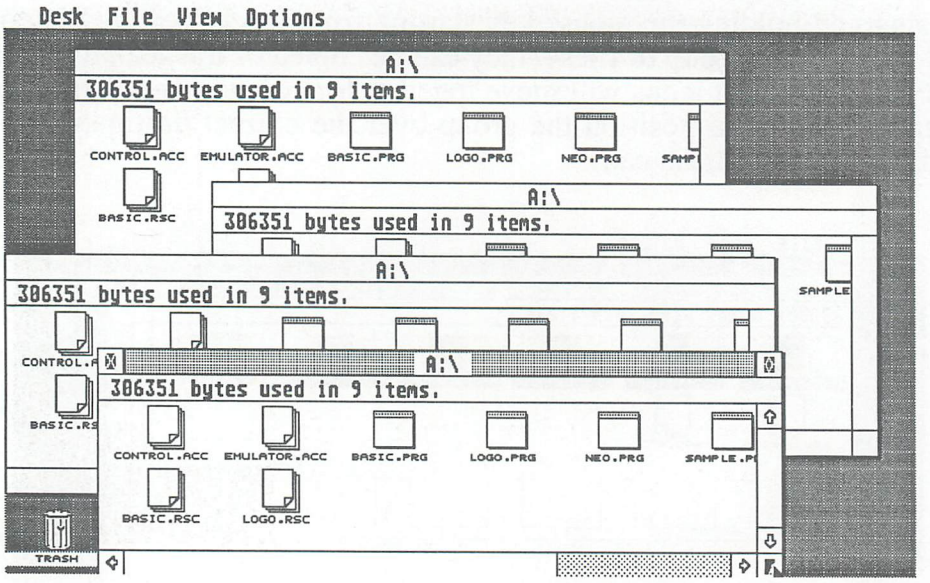


You can also select a group of files by holding down the <Shift> key and clicking each file. All files will move together for copying or deletion.

2.5 Multiple windows

Until now we have only opened one window on the screen, to display the current disk directory. But the GEM operating system lets you open up to four windows at once.

Double-click disk icon A four times. This gives you four disk directories.

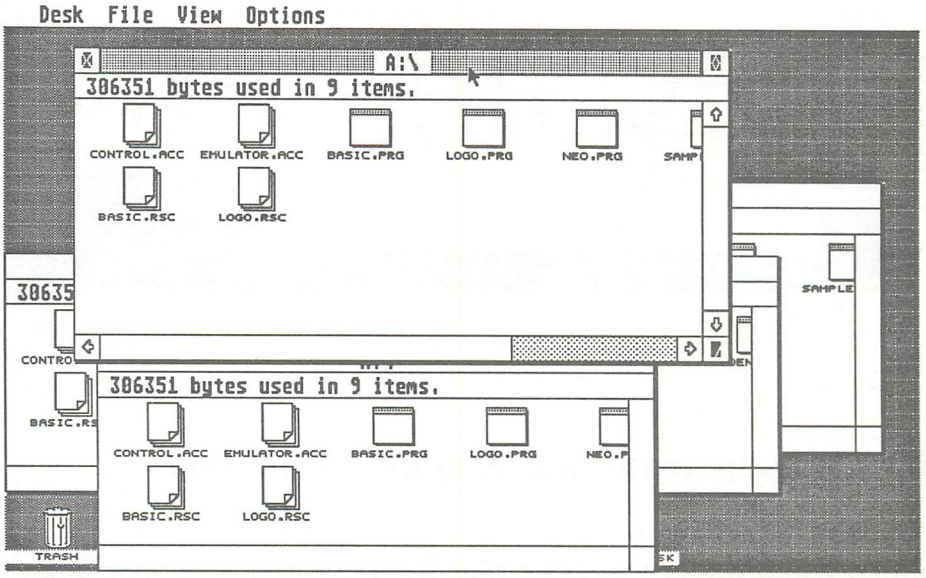


Your screen will look crowded. The ST has stacked the four directories as if they were four sheets of paper piled atop one another. The window margins of the top page are visible (e.g., close box, move bar, etc.), while the other three pages have no margins. It's possible to change the size of the current directory using the size box. We can also move the directory to another position, or close it.

As you probably already found out, your ST remembers all the information contained in the hidden (or partially hidden) disk directories. If you move the current directory, the hidden pages become partially or completely visible.

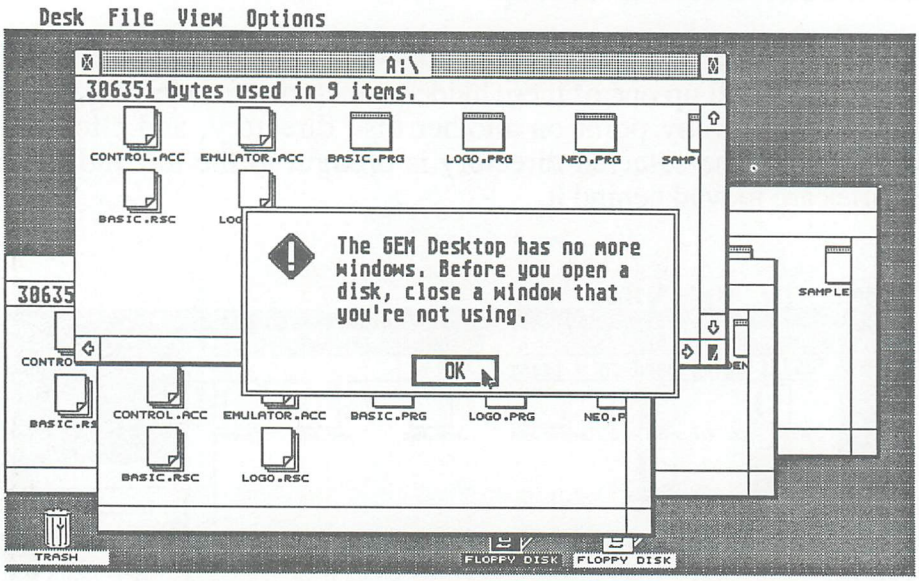
2.5.1 From bottom to top

You can easily call up one of these hidden directories for viewing. Move the mouse pointer to any point on another disk directory, and click the left mouse button. The selected directory is brought to the top and the other directories are moved behind it.



The ST can have only four windows open at once. If you attempt to open a fifth window, you will get an alert message. One of the windows must be closed before you open another. Click OK.

You may not need four windows at once, but some applications make use of multiple windows. There are ST programs on the market that use four windows for calculations (***PowerLedger***), data management (***DataRetrieve***), and graphics (***PaintPro***).



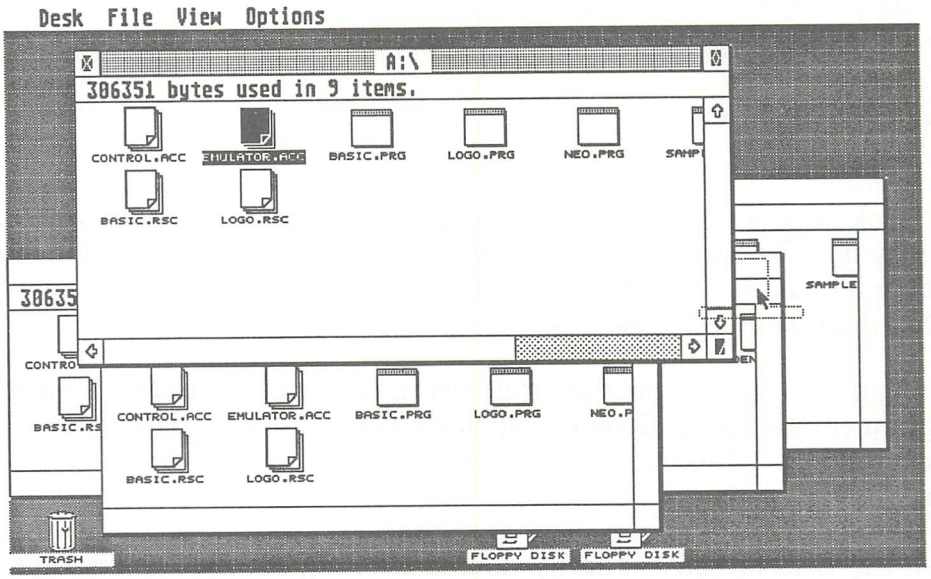
2.5.2 Copying from one window to another

Multiple windows let you copy files from one disk directory to another. This is useful for copying files into folders on the same diskette. One window shows the main directory while the other window shows the contents of the folder.

If you only have a single disk drive, this one drive acts as drives A and B. This means that you will have to exchange disks when prompted to do so by the system.

However, if you own two disk drives, you simply put the source disk into drive A and the destination disk into drive B—the ST does the rest. A hard disk or RAM disk is represented by drive C. Multiple windows help this process.

2.6 Icons



We have already mentioned *icons*. There are three types of file icons: the folder, the block with the darkened top border, and the pile of paper with the corner of the top sheet folded over. We have already worked with the folder icon, so we'll discuss the other two file icons now.

2.6.1 Programs

The block with the dark upper border has one of three file extensions: `.PRG` (program), `.TOS` (Tramiel Operating System) or `.TTP` (TOS takes parameters). This means that the program either operates under GEM (up to 4 windows and graphic interface: `.PRG`), or without GEM (`.TOS` or `.TTP`).

Double-click the mouse on a `.PRG` program icon to open a program. The screen turns dark, the pointer turns into a busy bee, and the program loads

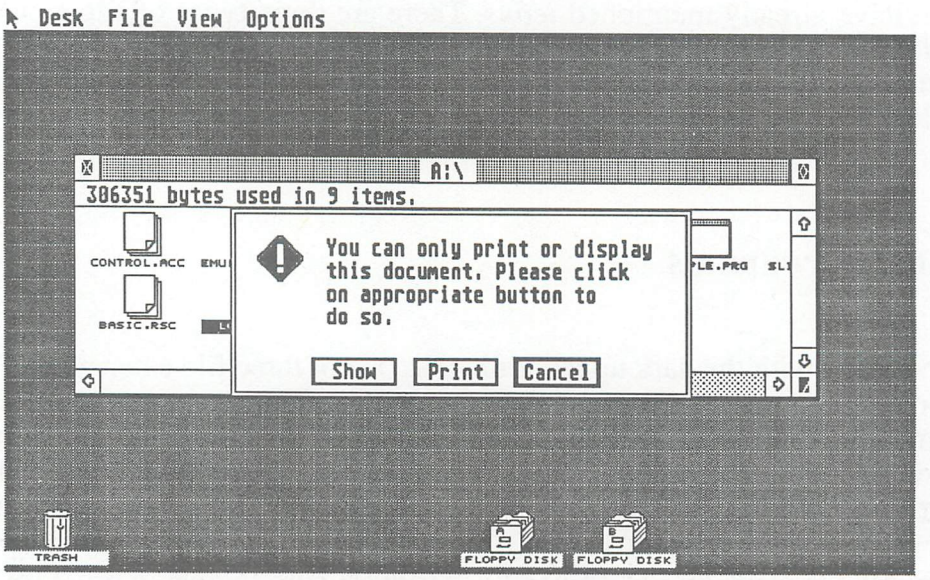
into memory. After a few seconds, the opening screen of the program appears. Programs with .TOS extensions will clear the screen when they are opened. Programs with .TTP (TOS takes parameters) extensions will first display a dialog box requesting input of parameters; these parameters vary depending on the program.

2.6.2 Data files

The stack of papers with the top sheet folded represents a *data file*. A data file may be a BASIC or LOGO program file (i.e., a program created in one of these languages, and not necessarily from the operating system), a word processing file, or a database file.

Files can also be complex subroutines, automatically loaded by a main program (block icon). Screen graphics can be stored as data files. These files have a .PIC extension, and are recognized by the program used to create them.

Open the BASIC.RSC file by double-clicking it. The ST displays the following information:



If the master program was not available with the file, the file is not loaded. You could CANCEL the procedure or Show the file. If you click Show the first line of your screen would look like this:

```
Td$vr (Desk File Run Edit Debug About ST BASIC---------------- De
```

This file is read directly after a matching main program (BASIC.PRG) is loaded. The ST can make direct use of this material—all we can do is see it.

The next example is a text file. If the program had been shorter than the screen, ----END OF FILE---- would have appeared on the screen instead of -MORE-. To see more of the file, you would press <Enter> or <Return> to see the next program line. Press the spacebar to display the next screen of the file. Other files that can be viewed like this are usually BASIC programs (.BAS extension), LOGO programs (.LOG extension) or word processing files (mostly .DOC or .TXT extensions).

```
                    1ST WORD RELEASE NOTE

                        VERSION 1.02

    This disk contains 1st Word, the GEM word processor written by
    GST of Cambridge, England and supplied with your Atari ST
    computer.

    In order to provide you with a fully working word processor as
    soon as possible, the 1st Word User Guide has been supplied on
    the microfloppy disk together with the 1st Word software.

                        Backing up

    Before you do anything else, backup the 1st Word disk.

                    Viewing the User Guide

    -MORE-
```

When the words ----END OF FILE---- appear on the screen, press any key to return to the Desktop. If there is no immediate end to the data file being observed, press <Control> and <C> at the same time to stop the display. The disk directory should reappear on your screen.

2.7 The keyboard

You're probably getting anxious to start programming. Let's have a close look at the keyboard first.

You'll remember that the ST can communicate in different *programming languages* such as BASIC, LOGO and C. However, these programming languages are not built into the ST. We have to load these languages from the Language Diskette.

Insert the Language Diskette in a disk drive, double-click the disk drive icon, then double-click the `BASIC.PRG` icon. The screen displays four windows. We'll use these to test the ST keyboard.

Most ST keys are not used with every program since the ST can function without the keyboard. This is why the mouse, icons and drop-down menus exist.

Once BASIC is loaded, we can enter data from the keyboard. The entries appear in the command window at the bottom left.

Your ST keyboard has four sections:

1) QWERTY typewriter keyboard
2) function keys
3) editing keys
4) numeric keypad

2.7.1 Typewriter keyboard

Here you have all the upper and lower case letters of the alphabet, as well as many special symbols. Upper case letters are entered by pressing <Shift> and the desired letter. Like a typewriter, there are special keys to perform different functions.

2.7.1.1 <Alternate>

Pressing <Alternate> in conjunction with other keys including <Shift> <Alternate> allows ST users in different countries to access the special characters required by their language not found on a normal QWERTY keyboard. This gives them special characters such as å or ß, normally not accessible. Programmers can also take advantage of these keys. A programmer could increase the number of the function key functions from 10 to 30 by using the <Alternate> and <Shift><Alternate> keys.

2.7.1.2 <Control>

Some of the keys are used in conjunction with the <Control> key to perform special functions. For example, a <Control><C> in LOGO or <Control><G> in BASIC stops a program during execution.

2.7.1.3 <Shift>

Pressing a letter key while holding down <Shift> displays the upper case letter, or the character that appears on the upper half of the key (i.e., <Shift><3> will print a "#").

2.7.1.4 <Caps Lock>

The <Caps Lock> key is essentially an on/off switch. Press it once to type upper case letters without having to hold the <Shift> key. Press <Caps Lock> a second time and the keyboard returns to its normal lower case mode.

2.7.1.5 <Tab>

The <Tab> key is a *tabulator*, which moves the cursor a specified number of spaces to the right in certain application programs. The cursor is shown as either a block or vertical line. <Tab> is handy for formatting numbers or text.

2.7.1.6 <Esc>

<Esc> (Escape) is similar to <Backspace> in many programs. It returns you to the main menu in some word processing programs. In other programs, <Esc> frequently serves as a program interrupt.

GEM uses the <Esc> key for two functions. First, you may update a disk directory when you switch diskettes by pressing <Esc>. Second, you can change a disk name in `Info` by pressing the <Esc> key to erase the old name.

2.7.1.7 <Backspace>

This key moves the cursor one space to the left, erasing the character to the left. Holding the key down will repeat the action (i.e., the cursor will continue moving to the left until the key is released).

2.7.1.8 <Delete>

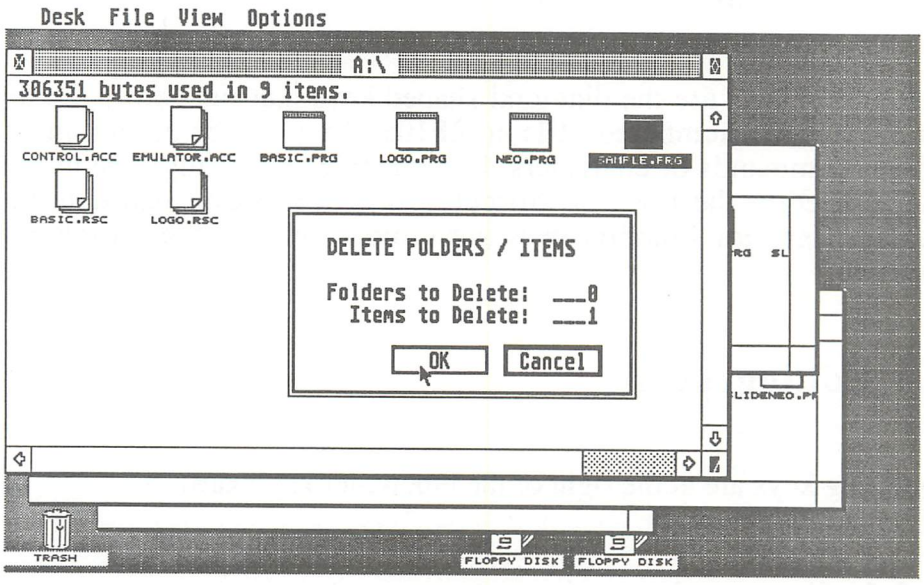
The <Delete> key erases the character on which the cursor rests. <Delete> and <Backspace> have the same functions in most programs.

2.7.1.9 <Return>

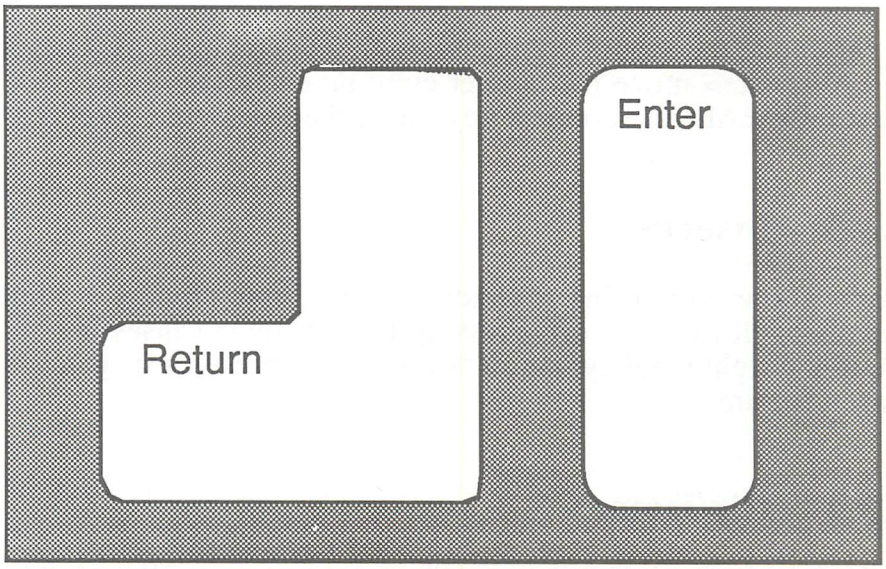
The <Return> key has its origins in the days of the typewriter, when the return lever performed a carriage return and line feed. Even if you aren't using a printer, the ST still displays data on the screen. Pressing <Return> advances the cursor one line down, and to the left margin of the next line. Languages such as BASIC use <Return> to enter the current program line.

The <Enter> key often has the same function as <Return>. Both keys are called data entry keys in this book.

GEM uses <Return> to confirm dialog box information, if you prefer to input through the keyboard instead of the mouse. When a dialog box gives you two choices, such as `OK` or `Cancel`, you will notice that the border of one of the buttons is wider than the other. Pressing the <Return> key is the same as selecting the enlarged button.



Press <Return> to cancel



Data entry keys — <Return> and <Enter>

2.7.2 The function keys

The function keys are the diamond-shaped keys that run across the top of your ST, and are numbered <F1> to <F10>. They can be programmed for different commands or characters. For example, <F2> might be used in a game to specify the level of difficulty, or <F1> might load data into a database. Uses for function keys are limited only by the programmer's preference.

2.7.3 Editing keys

The editing keys are at the right of the typewriter keyboard:

2.7.3.1 Cursor keys (<→><↑><←><↓>)

The cursor keys move the cursor in the direction of their marked arrows.

2.7.3.2 <Shift><Alternate> cursor keys

You can move the mouse pointer using these combinations. Cursor keys with <Alternate> move the pointer eight pixels in the desired direction, while <Shift><Alternate> cursor keys move the pointer one pixel at a time.

2.7.3.3 <Insert>

<Insert> allows you to insert words, text, etc., into already existing text. <Insert> toggles on and off. Pressing the key once causes the cursor to move to the right when typing in new text, and pressing it again turns off the insert feature.

2.7.3.4 <Clr/Home>

<Clr/Home> has a double function in some programs. Pressing this key sends the cursor to the "home" area (the upper left corner of the screen). A <Shift><Clr/Home> clears the screen completely.

2.7.3.5 <Help>

This key often does just what its name implies. If you don't know what to do next, press <Help> for advice.

2.7.3.6 <Undo>

We read about <Undo> earlier when reading about the VT52 emulator. As the name suggests, <Undo> undoes (cancels) the previous entry.

2.7.3.7 <Alternate><Help>

<Alternate><Help> sends a "snapshot" of your screen to a properly-connected printer (as discussed in section 2.3.2.4). Pressing <Alternate><Help> a second time turns off the screen dump.

Note: If you don't have a printer connected, your ST may crash (stop operating) when you press <Alternate><Help>. The only way out of a crash is to press the reset button in the back, or switch the system off and on again.

2.7.4 Numeric keypad

2.7.4.1 <Enter>

The <Enter> key has the same function as <Return>, as mentioned above.

2.7.4.2 Numbers, math symbols and decimal point

The ten digits and arithmetic symbols are the same as those on the top row of the keyboard, except the keypad has them arranged in the standard keypunch format for single-handed use.

Chapter 3

ST BASIC

ST BASIC

This chapter will introduce you to the BASIC programming language. BASIC stands for Beginner's All-purpose Symbolic Instruction Code. All we'll do here is familiarize you with some commands, help you write a few simple programs, and give you some hints on larger programs. Once you understand the essentials of BASIC, you should read other books on the language, such as the *ATARI ST BASIC Training Guide* from Abacus.

3.1 How the ST understands BASIC

3.1.1 What is BASIC?

Up until now, all we have done with our ST is answer questions and respond to commands. Now it's time for some real dialogue between you and your computer. To communicate, both parties must be able to speak the same language. The ST understands several languages—but these are *programming languages*, not French or German.

The Language Diskette contains everything the ST needs to be able to understand a language. This diskette contains the vocabularies and the grammar (or *syntax*) of two languages—BASIC and LOGO.

Once one of these languages is loaded into the ST's memory, you can use the keyboard to communicate with the ST. We'll start with BASIC.

3.1.2 Loading BASIC

Insert the Language Diskette into drive A. Simply open the directory of that diskette by clicking the drive's icon, then double-click the `BASIC.PRG` icon.

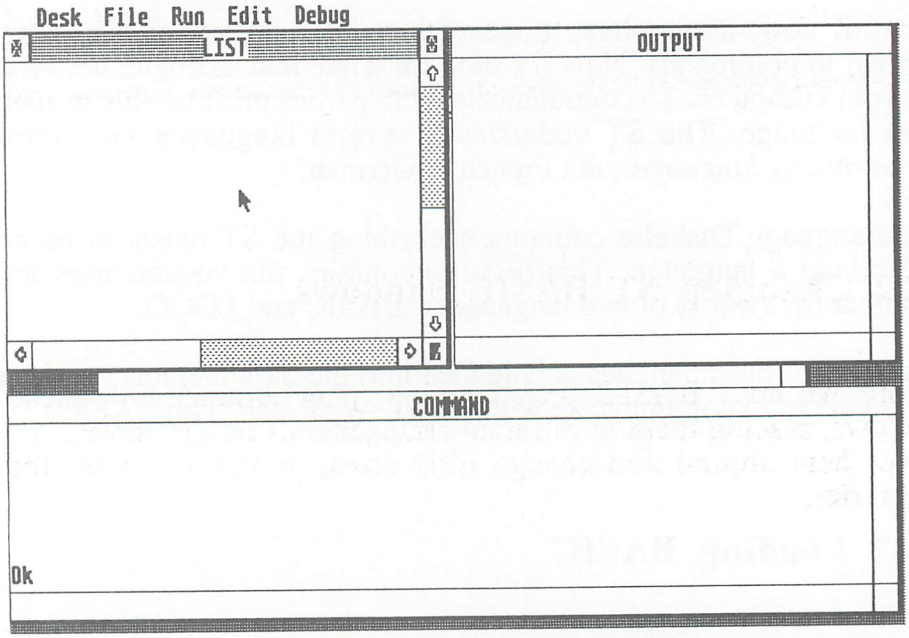
3.2 Windows in BASIC

3.2.1 How windows work in BASIC

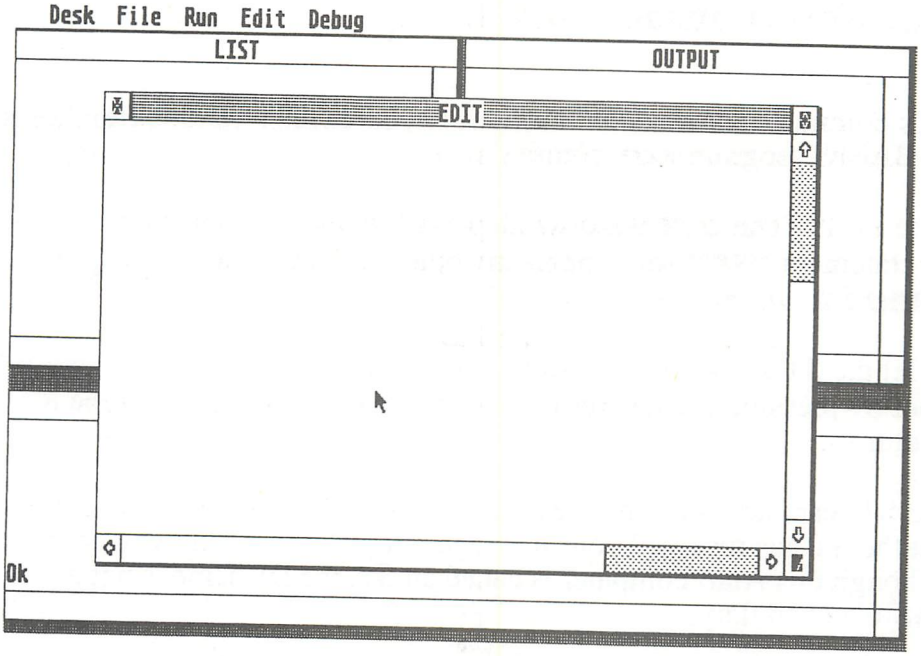
BASIC has altered the look of our screen considerably. The menu line at the top displays terms that are familiar to us, but four new windows appear:

- The **OUTPUT** window displays the program
- The **COMMAND** window accepts program data
- The **LIST** window lists the program code
- The **EDIT** window is used to edit the program

The **COMMAND** window can be used to call up the other windows. Type `List` on the keyboard and press the <Return> key. The **LIST** window appears after a brief time.



The **EDIT** window can be seen in the center of the screen and behind the other three windows. Move the pointer to that window, click it, and the **EDIT** window will become the top window.



3.2.2 Changing ST BASIC windows

Before we start BASIC programming, play around with these four windows, placing them in different arrangements on the screen. You can move them around and change their sizes, as you did with the disk directories.

3.3 Beginning with BASIC

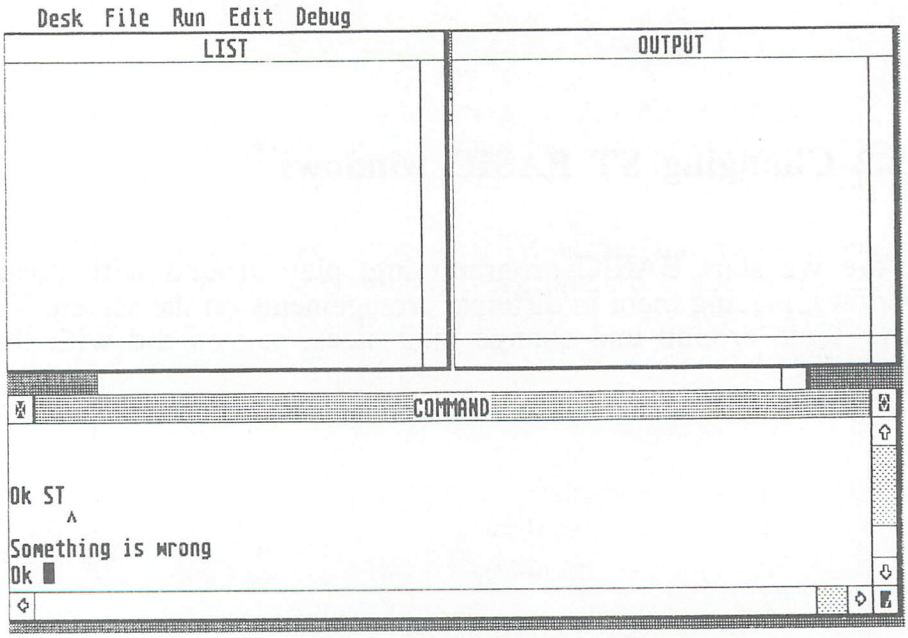
3.3.1 Direct mode

Let's communicate with ST BASIC. If you haven't yet double-clicked the ST BASIC program icon, please do so.

Type in `ST`. The **COMMAND** window will display the letters `ST` following the statement `OK`. If you type in any other text, the letters will appear in the **COMMAND** window.

To let the ST know that we want to do something, we must end any BASIC input by pressing the <Return> or <Enter> key. So after you type `ST`, press <Return>.

The ST responds with the message `Something is wrong`. This means that the ST has received your message—it just doesn't know what it means. Although this Atari computer is called an ST, the ST name is not a part of its BASIC vocabulary.



If you try to type in other words or phrases, the response will be the same: The text, an error message, and OK. The OK indicates that the ST is ready for more data.

Let's try something else. Enter the following command:

```
PRINT "atari st" <Return>
```

You should see `atari st` displayed in the **OUTPUT** window.

This output was initiated by the `PRINT` command, which is in the ST BASIC vocabulary. You can put anything between those quotes instead of `atari st`: your name, your mother's name, the first line of War and Peace, etc. The ST executes your command when you press <Return>.

This procedure is called *direct input*. Executing commands outside of a program is called *direct mode*.

3.3.2 Program mode

Direct mode requires you to constantly type in line after line. If we want to execute a number of tasks, we have to write a *program*.

BASIC was invented for programming. Once you write and enter a program, tasks can be carried out as often as you desire, started with a single command. This is *program mode*.

Let's see how we can get the ST to work for us, using the most important BASIC commands and techniques. For example, clear (erase) the second window (**OUTPUT**) by typing the following line. (If you make an error, <Backspace> over the mistake and retype it):

```
clearw 2

clear  =  clear
w      =  window
2      =  window number
```

`clearw 3` clears the **COMMAND** window.

3.4 The BASIC vocabulary

3.4.1 PRINT and INPUT

3.4.1.1 Line numbers

Each line in ST BASIC must have a line number. Do not type in a command after OK—start with a number. The best way to do it is to start with 10 and increase by steps of 10 (10, 20, 30, etc.). This gives you extra space for inserting other lines as they are needed (15, 17, etc.).

The BASIC command or commands follow the line number.

3.4.1.2 PRINT

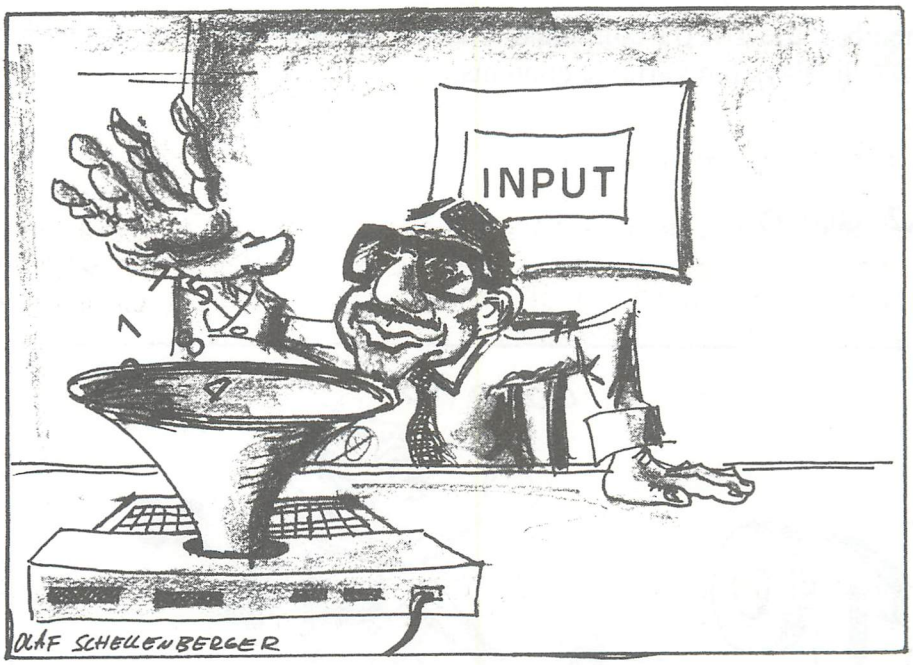
As we have already seen, PRINT writes characters, known as a *string*, on the screen. Let's use our earlier example in a program line:

```
10 PRINT "Atari ST"
```

If you make an error, <Backspace> over the mistake and retype it. Press <Return> to enter the line.

Our program is now in memory, but has to be executed. BASIC programs are started with RUN. You can enter this command in direct mode, or click Run from the menu bar. The program then runs—Atari ST appears on the screen in the output window. We have the same result as before, with an important difference: **Direct mode is typed in directly, while programs require line numbers.**

3.4.1.3 INPUT



Clear the **OUTPUT** and **COMMAND** windows by typing `clearw 2` and `clearw 3` in the **COMMAND** window. Our one-line program is in memory, but is invisible for now. Clear the program from memory by typing the direct command `NEW`. Now we want the ST to print what is between quotes, and then wait for input from the keyboard. We can input text in a program with `INPUT`:

```
10 input "What should I write?"; a$
20 print a$
```

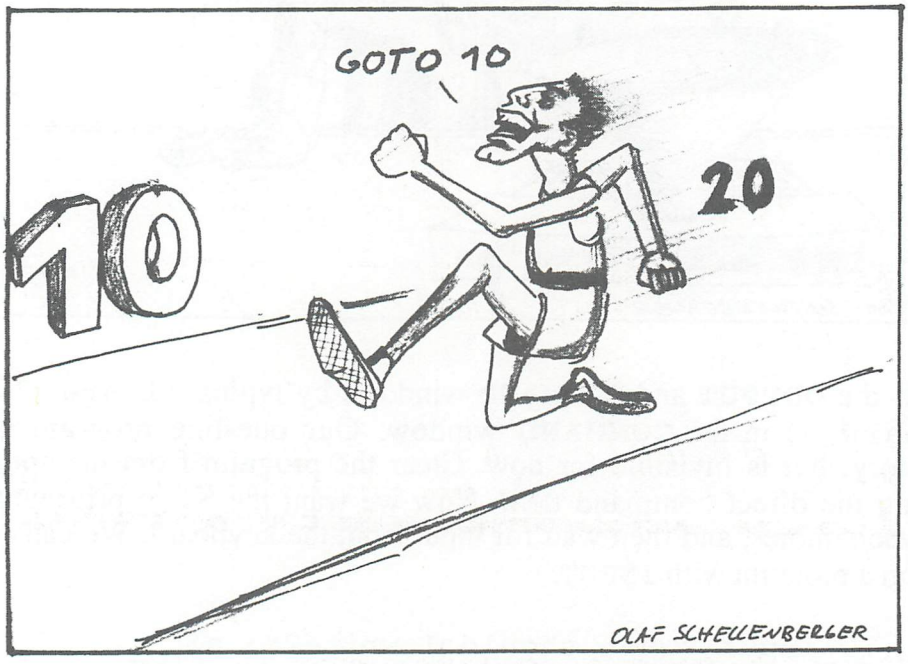
Now click `Run`. The ST asks you `What should I write?`. Answer by typing in some text. Press <Return> to see this text.

Line 10 prints the question and a question mark on the screen. This is done by using the `INPUT` statement, which expects a response. Your response is stored as a *variable* called `a$`. The dollar sign signifies that the answer was in the form of text.

We would have to remove the dollar sign to accept numeric values. Line 20 displays the contents of a$ (your entered text). The program ends, and OK appears in the **COMMAND** window.

Remember: the INPUT command relates a variable to a string or number. PRINT displays the variable contents.

3.4.2 GOTO



Erase program memory with NEW, and clear the three windows with clearw 1, clearw 2 and clearw 3.

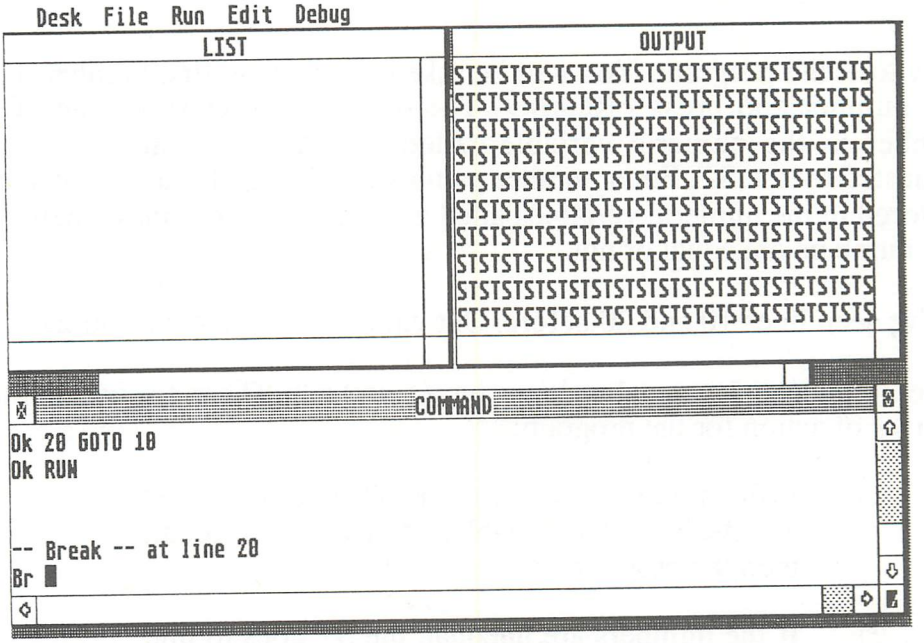
One of the simplest programs consists of two lines. Enter these lines on your ST:

```
10 print "ST";
20 goto 10
```

Now RUN this program. The string ST appears again and again. This will continue until the power is turned off, the reset button is pressed, or the <Control><G> key combination is pressed. A message will appear on the screen: Break at line xx. The program can be continued by typing CONT (CONTinue).

Line 10 PRINT displays ST on the screen. The semicolon at the end of the line causes the STs to be printed right next to each other.

Line 20 GOTO10 returns the program to line 10 (an *infinite loop*).



GOTO causes the program to jump to the specified line number.

3.4.3 IF . . THEN . . ELSE

GOTO is executed without any special requirements. An IF . . THEN statement can initiate a jump, but only after certain prerequisites are (or are not) fulfilled. Erase the ST's program memory by typing NEW and clear the

three BASIC windows as you did above, then enter the following program. If you make an error, <Backspace> over the mistake and retype it. Don't forget to press <Return> after each line.

```
10 input "1st number";z1
20 input "2nd number";z2
30 if z1=z2 then 60 else 40
40 print "The numbers are not the same."
50 goto 70
60 print"The numbers are equal."
70 end
```

Now RUN the program. You will be asked to enter the first number. Enter any number and press <Return>. The program stores the value of this number in the numeric variable called z1. A second number is then requested by the ST. Enter another number and press <Return>. This value is stored in the numeric variable called z2. The ST determines whether the two numbers are equal or not.

We've seen lines 10 and 20 before. Our variables here are z1 and z2.

Line 30 checks for equality between z1 and z2. There are two possible courses of action for the program:

1) If the numbers are equal, the ST goes to line 60 and declares that variables z1 and z2 are equal, then the program ends at line 70.

2) If the numbers are unequal, the ST goes to line 40. The IF...THEN...ELSE command states that IF something is fulfilled THEN a jump to the stated line occurs, or ELSE the alternate line number is executed. IF z1=z2 THEN go to line 60; or ELSE, go to line 40, which states that z1 and z2 are different. The program then goes to line 70, and the program ends.

Remember: IF...THEN...ELSE "jumps" to other parts of the program under certain conditions.

3.4.4 FOR...NEXT loops

You can write a program that counts while running, then stops at a predetermined number. To accomplish this, you need to write a *loop*. We can start with the same program presently in memory. Do not type NEW, but erase the three windows with clearw. Then enter the following four lines:

```
5   input "How many times";d
6   for i=1 to d
70  next i
80  end
```

To check how a program looks before you RUN it, move the mouse pointer to the drop-down menu **Edit** and click List. Each line of the program appears in the **LIST** window:

```
5   input "How many times"; d
6   for i=1 to d
10  input "1st number";z1
20  input "2nd number";z2
30  if z1=z2 then 60 else 40
40  print "The numbers are not equal."
50  goto 70
60  print "The numbers are equal."
70  next i
80  end
```

We added lines 5 and 6 to the program. We changed line 70 as well: the ST erased the old line 70 and put the new line 70 into memory. Previously line 70 was simply:

```
70  end
```

Line 70 now reads:

```
70  next i
```

We also added line 80. See how simple it is to edit the programs? You only have to enter the new information. The rest of the text does not have to be retyped, because it is still in memory.

What did we change? Line 5 is obvious. The variable d has the value of the number, indicating how many times the program has to run through. Then line 6 follows. Here is the beginning of the loop. The end of the loop is found in line 70. Everything in between belongs to the loop.

Let's follow the program:

Line 5: Number of times in variable d.

Line 6: Begin loop. The variable i (also called the *counter*) is raised by the value 1. Then the ST checks to see if i equals d.

Lines 10 to 60: Program like the one you wrote earlier.

Line 70: Jump to the beginning of the loop (line 6). There i is again raised by 1 and i is compared for equality to d. If this is not the case, the loop is performed until i equals d. When this occurs, the program jumps to line 80 (end of the program).

3.4.5 GOSUB...RETURN

GOSUB is an abbreviation for GOto SUBroutine. Subroutines are program segments that are called repeatedly. The main difference between GOTO and GOSUB is that GOSUB requires a statement to a) mark the end of the subroutine, and b) send the system back to the main program from which the GOSUB occurred. In reference to our two-line program, this would mean that the screen output of the string atari st is a *subroutine*, limited by the main program:

```
10  gosub 100
20  goto  10
100 print "ST";
110 return
```

The RETURN command in line 110 causes BASIC to return to the line following the GOSUB, and program execution continues from line 20.

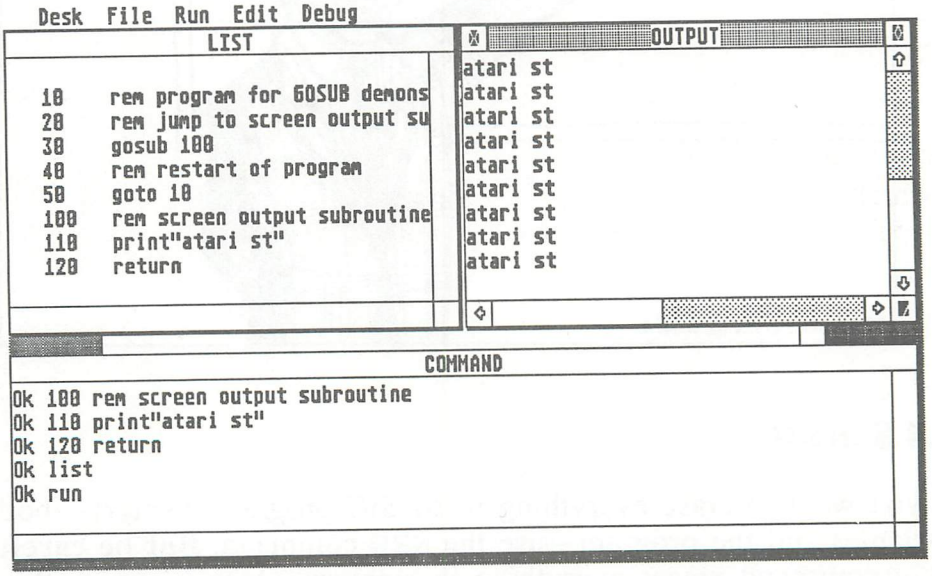
Maybe we haven't clearly defined the subroutine here. But in Section 3.9, we'll see how well subroutines facilitate program control.

3.4.6 REM statements

If you type the word REM (abbreviation for REMark) at the beginning of a line, your ST ignores this line and continues with the next. We can enter comments following the REM which remind us and inform others of how the program operates. This is especially valuable in longer programs. Let's add some REM statements to our GOSUB program:

```
10   rem program for GOSUB demonstration
20   rem jump to screen output subroutine
30   gosub 100
40   rem go back to start of program
50   goto 10
100  rem screen output subroutine
110  print "atari st"
120  return
```

Before you start the program with RUN, look at the listing by clicking List. The advantage of using REMs is that you can read this program months later, yet still be able to understand what the program is doing at a specific line, even if you have long forgotten the program.



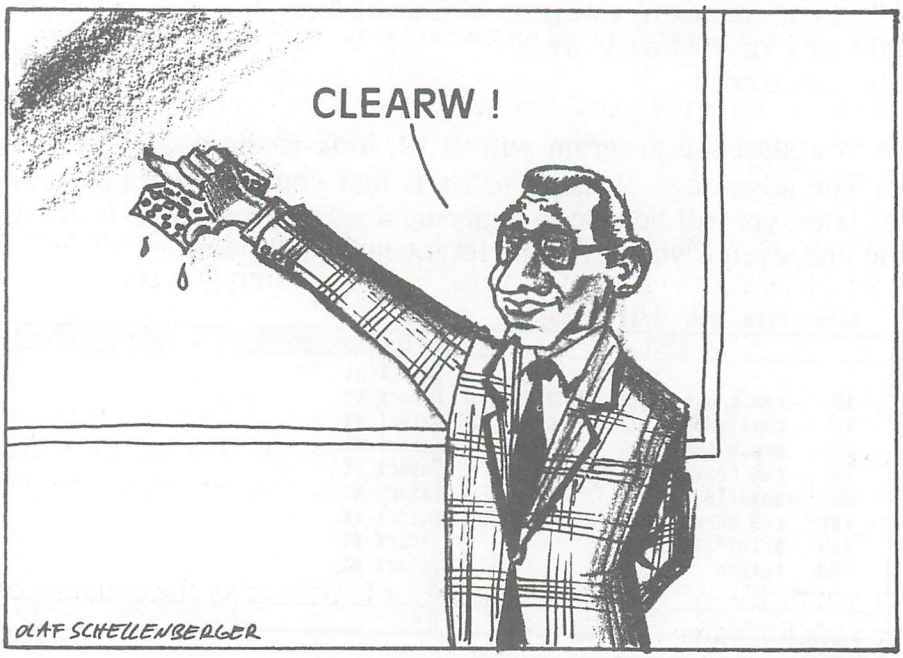
3.4.7 Clearing the screen

3.4.7.1 Erasing the windows: CLEARW

The command `clearw n` should be used before starting a new program.

The letter *n* can be values from 0 to 3, for the four ST BASIC windows:

n =0: Erase the **EDIT** window
n =1: Erase the **LIST** window
n =2: Erase the **OUTPUT** window
n =3: Erase the **COMMAND** window



3.4.8 NEW

If you want to erase everything in BASIC program memory—both the variables and the program—use the NEW command. **But be careful.** A NEW command erases everything in memory. Once you give the NEW command, your program is lost forever, unless it was previously saved onto a diskette.

3.5 Variables

You might have already noticed that programming a computer is almost impossible without the use of *variables*. Let's begin our explanation by discussing *numeric variables*.

3.5.1 Simple numeric variables

Your ST has to retain a lot of numbers in memory. To memorize the different numbers, the ST establishes a "drawer" in memory and asks you to name it. Until now we selected different names (`a`, `A$`, `z1`, etc.). For now we'll use the name `ST`, which becomes our variable name.

Up to now, we have assigned only one value to each variable when we used the `INPUT` command. But this can be simplified by establishing an equation.

For example, if we want to make our ST understand that the drawer with the name `ST` contains the number value 6, then we simply type:

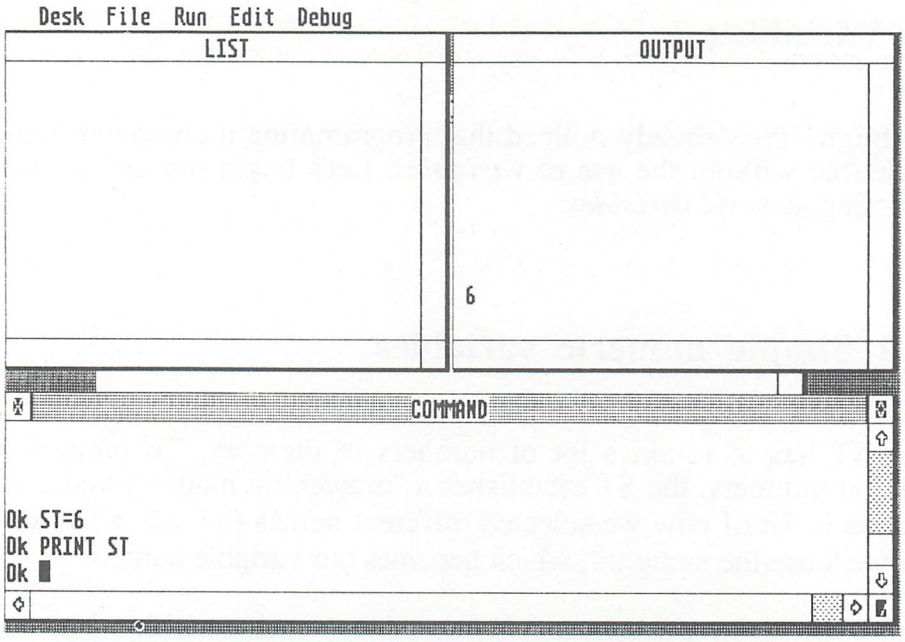
```
ST = 6
```

Don't forget to press <Return>. An error message does not appear in the **COMMAND** window. We know that everything is fine because the `OK` appears on the screen. Our ST remembers that the drawer called `ST` contains the value 6!

See for yourself—`print` is the magic word. To display the contents of the drawer with the name `ST`, we type:

```
print ST
```

If you followed the instructions step by step, the **OUTPUT** window should now display the following:



Maybe you're still unsure. In the previous line you typed `ST = 6`. The ST could have done that on its own. To prove that the variable is safely in the ST's memory, simply clear the screen with the commands `CLEARW 2` (for the **OUTPUT** window) and `CLEARW 3` (for the **COMMAND** window). Then repeat the command:

```
print ST
```

Again the number 6 is displayed. Now that you know how this works, give names and number values to other drawers:

```
Amanda    = 4
Brigitte  = 5
Claudia   = 6
Danielle  = 7
Rosalita  = 2.1
```

You can then request the ST to display these variables in the **OUTPUT** window with the `PRINT` command. There is enough free memory available in the ST to assign values for many thousands of entries.

A variable name may have a total of 31 characters. The first 31 characters are acknowledged by the ST. Any more characters are simply ignored.

There are two more things that you should know about selecting a variable name:

1. The variable name may not be a reserved BASIC word. For example, you can't name a variable `PRINT`. Try `print =5`.

   The error message `Something is wrong` appears, since you've used the ST's `PRINT` command as a variable name.

2. The first character of a variable name must be a letter (ST BASIC doesn't distinguish between upper case and lower case letters).

   For example, valid variable names include: `A3`, `Ac`, `B`, as well as `X`, `Y` and `C`. Variable names such as `4A` and `1E` are not permitted.

3.5.2 Integer and decimal variables

Now we'll assign different kinds of numerical variables to the ST. Let's test to see whether the high, multi-digit precision of the ST's computational output can be rounded off in order to simplify mathematical problems or numbers.

For example, enter the following line in the **COMMAND** window:

```
print 2/7
```

The slash represents the division of 2 by 7. The result of the division is displayed in the **OUTPUT** window: `.285714`. As you can see, the ST solved this problem to six decimal places.

One point to remember: when a result is less than 1, the ST does not show a zero in front of the decimal point. Like a pocket calculator, the ST displays numbers less than one as a period and decimal digits.

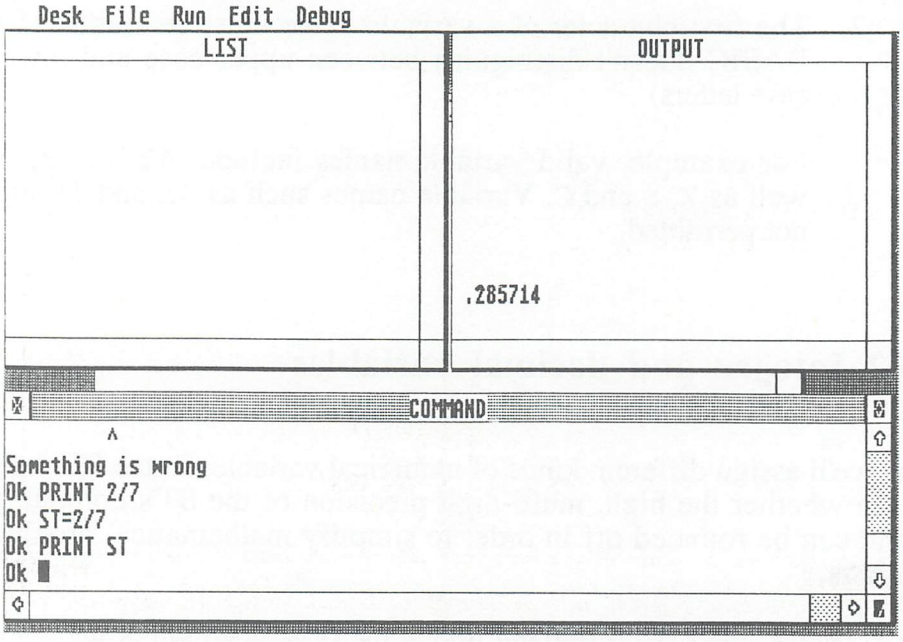
How can we store this calculation in one variable? Let's keep the name `ST` for a variable name and relate either the final result to it, digit for digit (in our case `ST=0.285714`), or enter the calculation itself:

```
ST=2/7
```

Now enter:

```
print ST
```

The following result appears in the **OUTPUT** window:



As you can imagine, a value with so many digits behind the decimal point requires large amounts of computer memory.

If you're not a mathematician and only want to calculate the multiplication table up to 25, then it would be inefficient to waste so much memory using decimal numbers. Furthermore, computers calculate much faster if they do not work with a lot of decimal places.

If rounded-off numbers are acceptable for the values you use, put a percent symbol (%) behind the variable name:

```
ST%=2/7
print ST%
```

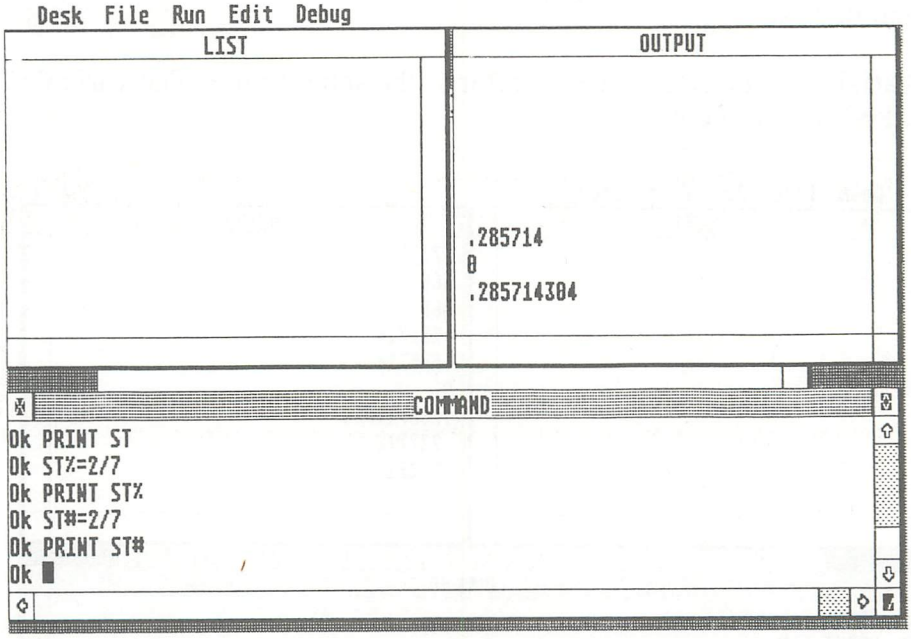
The result in the **OUTPUT** window is 0. Why? Because we decided to use integer variables which do not retain any fractional values.

On the other hand, if you need more than the 6 digits after the decimal point, enter a number symbol (#) behind the variable name:

```
ST# = 2/7
print ST#
```

Result:

```
.285714304
```



However, just remember that the results beyond the sixth digit are only approximate with this type of calculation.

In division problems, you should always consider which is more important: the amount of memory used in precise calculations, or the calculation speed.

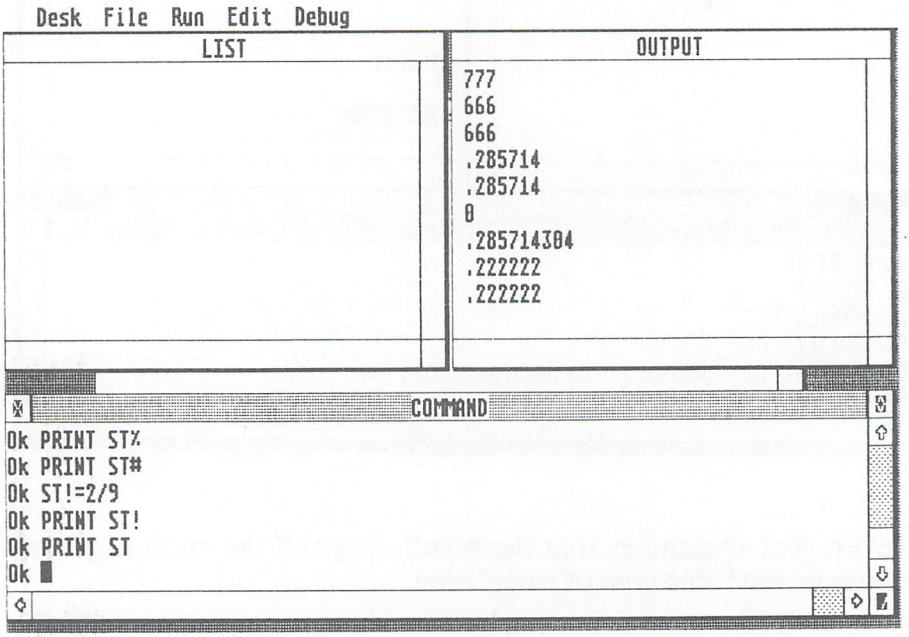
The advantage of having these three different precision indicators is that we can then use a variable name three times. Type these in:

```
          print ST
Result:   .285714
          print ST%
Result:   0
          print ST#
Result:   .285714304
```

When you're working with whole numbers, or *integers*, remember that integers may be no larger than 32767 and no smaller than -32767. Integer variables use the percent sign (%). Otherwise the ST memorizes negative numbers instead of positive numbers, and vice versa.

Finally, we can add an exclamation point (!) to the variable name. This symbol indicates that the calculations use simple precision math and six decimal places.

The variable names `ST` and `ST!` refer to the same figure. The variable `ST!` is marked for simple precision.



```
          ST!=2/9
          print ST!
Result:   .222222
          print ST
Result:   .222222
```

The variable ST! corresponds with the variable ST. Therefore, the contents of the variable ST changes and adopts the same value as the variable ST!.

3.5.3 String variables

You might be surprised that we handle *string variables*, that is, text, separately from the numbers. Is it possible to store text in our existing variable types? Let's try it. Store the word this in the variable ST. Type the following:

```
ST = this
```

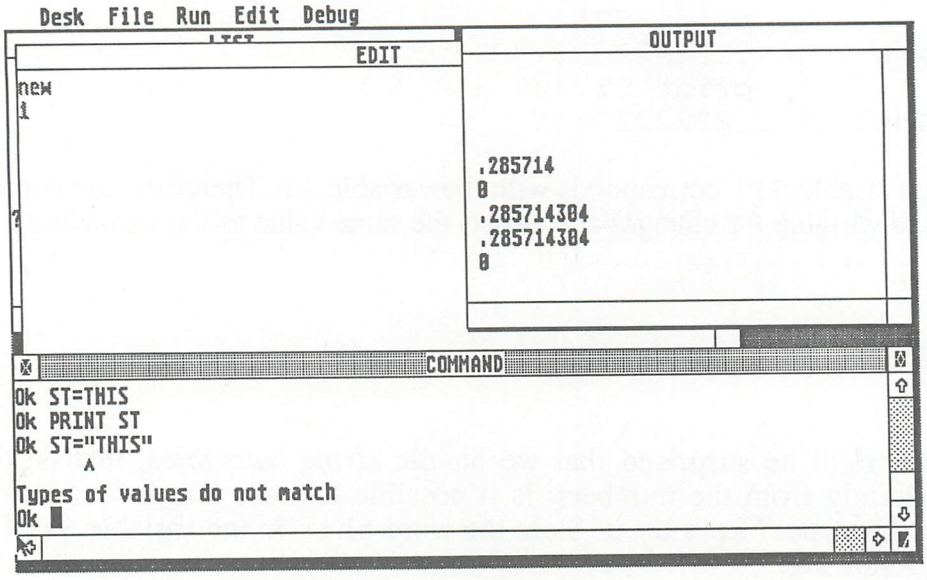
Now let's take a look at what is in variable ST:

```
          print ST
Result:   0
```

This is not what we wanted. The word this was accepted by the computer as another variable name. Therefore, the value of the "new" ST (until now=0) was simply copied into our existing variable ST. In effect, the "new" ST also became 0.

Remember what we said previously about the print command: if we want to output words or characters to the screen, we have to place the text between quotes. Let's try this with variables:

```
ST ="this"
```



The result is a new error message: `Types of values do not match`. This means that you used the wrong variable type. The numeric variable `ST` is not an acceptable variable type for storing text.

We need another symbol to designate a string variable: the dollar sign (`$`).

If we add this symbol to a variable name, then we create a string. You can store up to 255 characters in a string variable.

Type the following lines into the **COMMAND** window with the correct variable type, as a string variable:

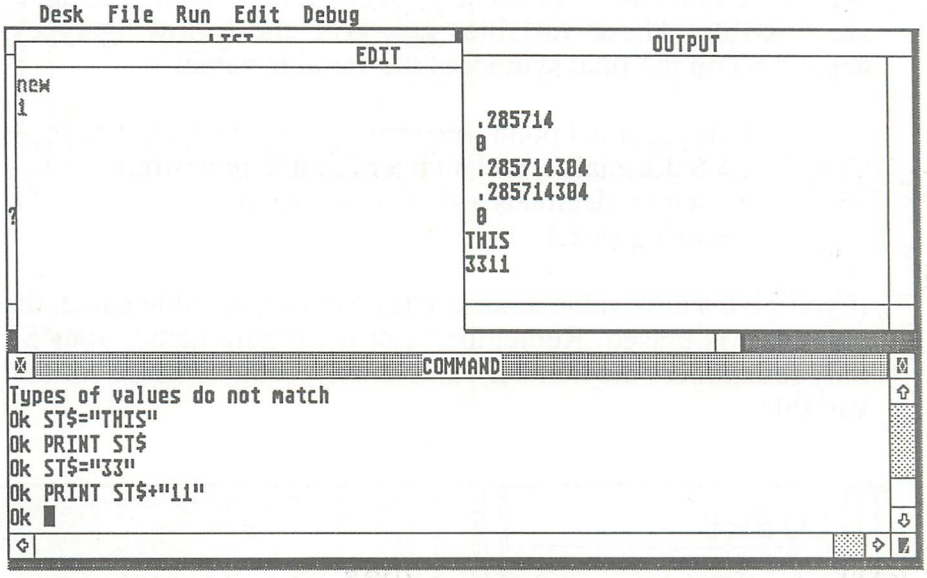
```
                ST$="this"
                print ST$
Result:         this
```

You may also store numbers in a string variable. You must enclose these in quotes as well, or the error message `Types of values do not match` reappears.

You cannot directly calculate with a string variable. Try it:

```
ST$="33"
print ST$+"11"
```

Result: 3311



In this case the values were not added together mathematically, but placed side by side. The string variable `ST$` with the text `33`, and the entered string text `11` were simply merged (or *concatenated*) into `3311`.

We found out that text variables may only contain up to 255 characters. This means that we would be unable to fit all those characters of a standard 8 1/2 x 11" page of text (which can hold about 2000 characters) into one string variable. However, we would be able to distribute the 2000 characters over at least 8 variables for storage.

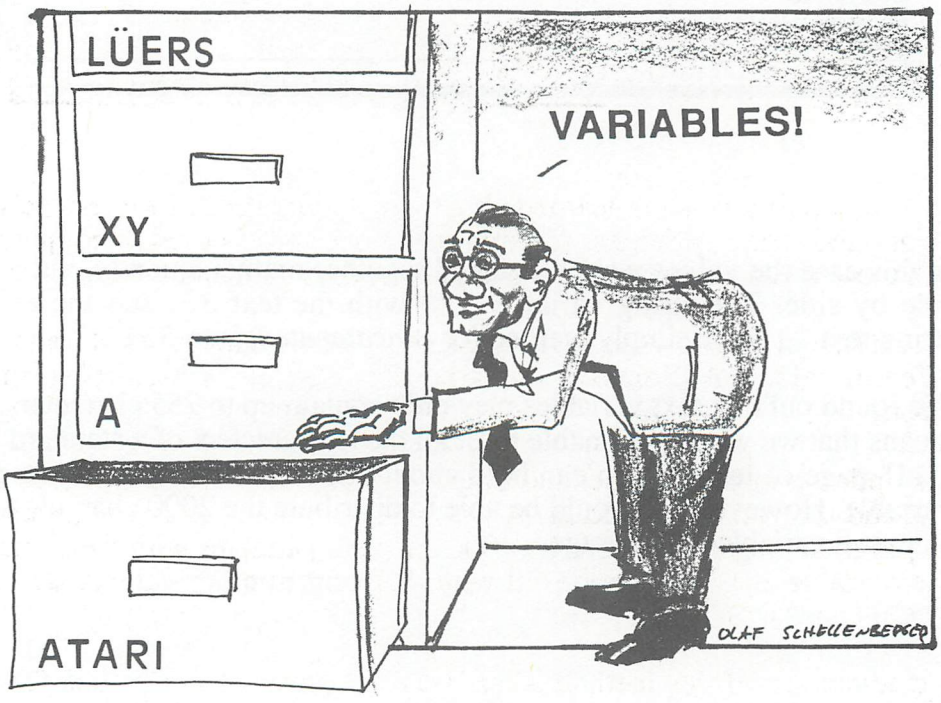
Two more notes about numerical and string variables:

1. If you enter the CLEAR command, all text and variable values are set to zero. Always put a CLEAR command at the beginning of a BASIC program.

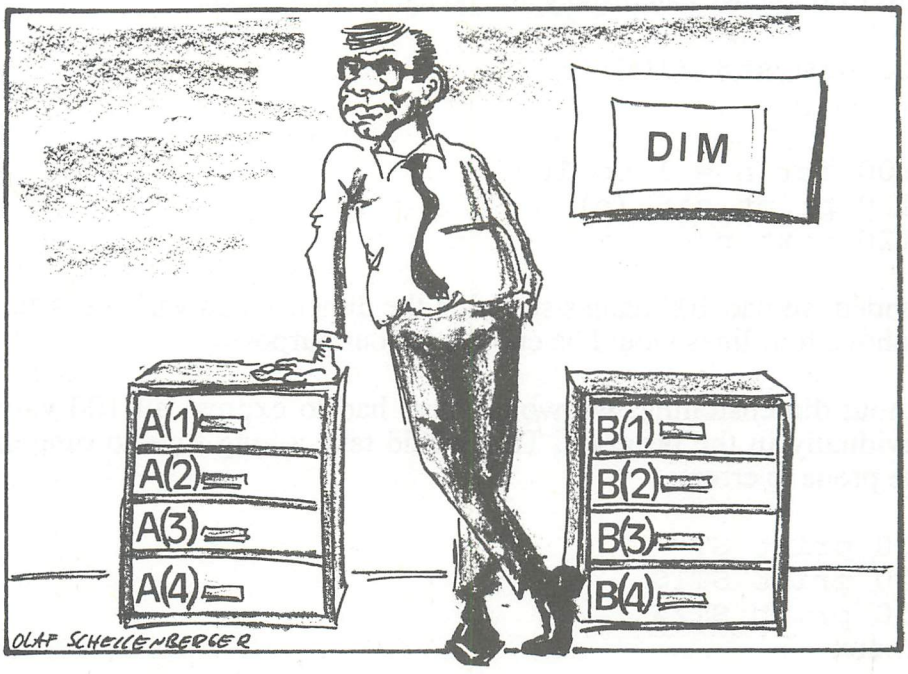
2. We learned that three different number values per variable name are possible. These variables can store the following types, depending on the final symbol of the variable name:

   %=no decimal point (integer)
   ! = 6 decimals = real number (simple precision)
   #= with 9 decimals = double precision
   $= string (text)

   If you give a new value to an already existing variable name, the old value is erased. Remember: For a variable name, your ST only remembers the number value most recently entered for that variable.



3.5.4 Dimensioning variables



Up to this point we have learned about double precision, simple precision, integer and string variables. An important command is used in conjunction with these types of variables—the dimensioning command `DIM`.

A `DIM` statement allows us to use the same variable name several times with different values. A dimensioned variable is always signified by using an *index*.

This index number is appended to the variable name in parentheses after `DIM` and the variable name. `DIM ST(1000)` entered at the beginning of the program means that there are 1000 different variables with the name `ST`. The variable `ST` can be indexed with different number values: `ST(1)`, `ST(2)...ST(1000)`.

The advantage of this method is that we don't always have to look for new variable names for certain program functions—we can retain the name of the original variable. Certain variable contents in a program could be called

repeatedly without having to redefine a variable name. For example, you could use a variable for the index figure, which you might increment (raise) with the help of a FOR...NEXT loop.

In this example we can display all 100 names to be stored:

```
10 dim st$ (100)
.
.
100 for n = 1 to 100
110 print st$ (n)
120 next n
```

Provided we had 100 names stored in the dimensioned variable ST$, then the above four lines would be enough for our purposes.

Without dimensioning, we would have had to express all 100 variables individually in the program. This would take a long time to run, and be more prone to errors:

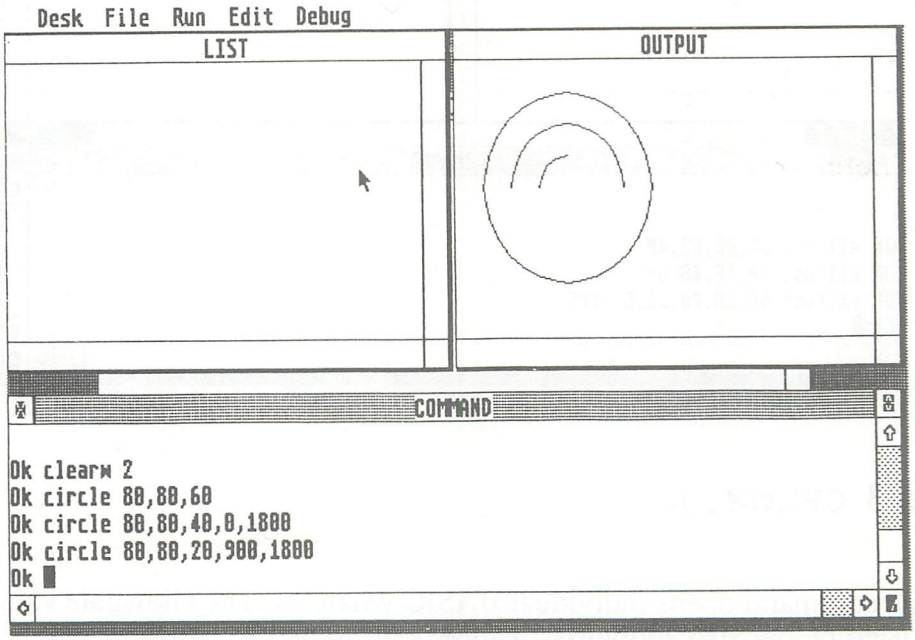
```
10 print ST
20 print ST1$
30 print ST2$
...etc.
```

3.6 ST BASIC graphic commands

We now know the most important BASIC commands. Before we move on to larger programs, here are some interesting graphic commands with which you should be familiar.

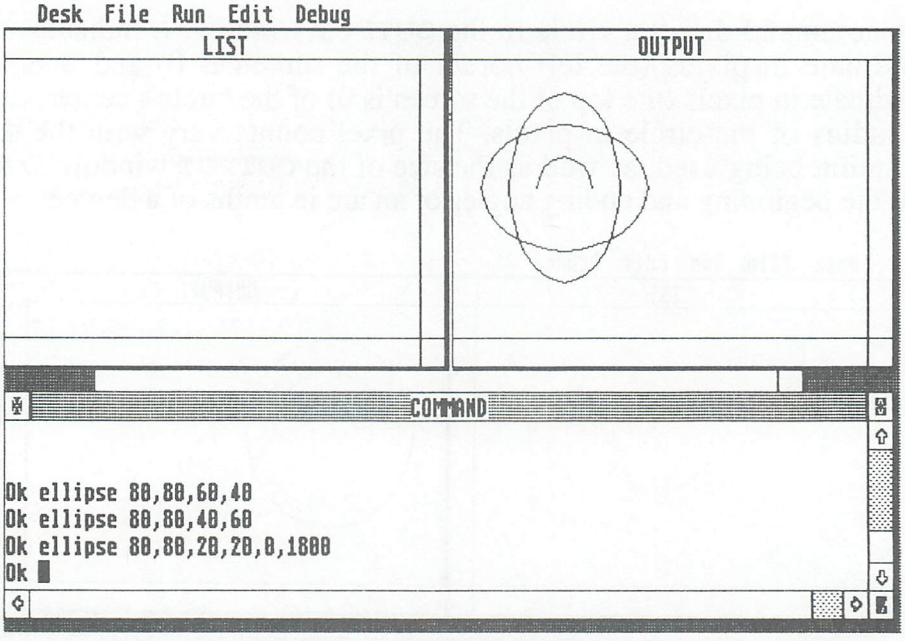
3.6.1 CIRCLE A,B,C,D,E

This command draws a circle in the **OUTPUT** window. A indicates the X coordinate in pixels (the left border of the screen is 0) and B is the Y coordinate in pixels (the top of the screen is 0) of the circle's center. C gives the radius of the circle in pixels. The pixel counts vary with the screen resolution being used, as well as the size of the **OUTPUT** window. D and E state the beginning and ending angles of an arc in tenths of a degree:



3.6.2 ELLIPSE A,B,C,D,E,F

This command draws an ellipse in the **OUTPUT** window. A indicates the X coordinate of the ellipse's center in pixels (the left border of the screen is 0) and B indicates the Y coordinate of the ellipse's center in pixels (the top border of the screen is 0). C is the X coordinate of the radius in pixels and D is the Y coordinate of the radius in pixels. E and F are the beginning and ending angles of the ellipse respectively in tenths of a degree. The pixel counts for the X and Y coordinates depend upon the resolution being used as well as the size of the **OUTPUT** window.



3.6.3 OPENW *n*

This command opens individual BASIC windows. The allowable values for *n* listed below also apply to CLEARW, FULLW and CLOSEW:

0=**EDIT** window
1=**LIST** window
2=**OUTPUT** window
3=**COMMAND** window

3.6.4 CLOSEW *n*

This command is used for closing individual windows:

0=**EDIT** window
1=**LIST** window
2=**OUTPUT** window
3=**COMMAND** window

3.6.5. FULLW *n*

This command opens individual BASIC windows to full screen size:

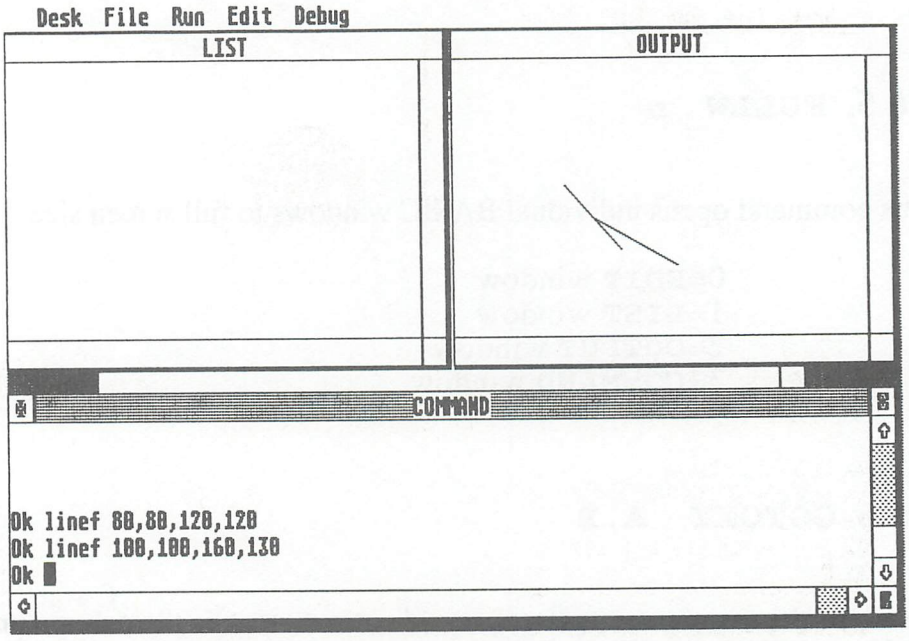
0=**EDIT** window
1=**LIST** window
2=**OUTPUT** window
3=**COMMAND** window

3.6.6 GOTOXY A,B

Positions the text cursor at screen coordinates A,B. A is the column position of the coordinates, and B is the row position. These numbers change with screen resolution.

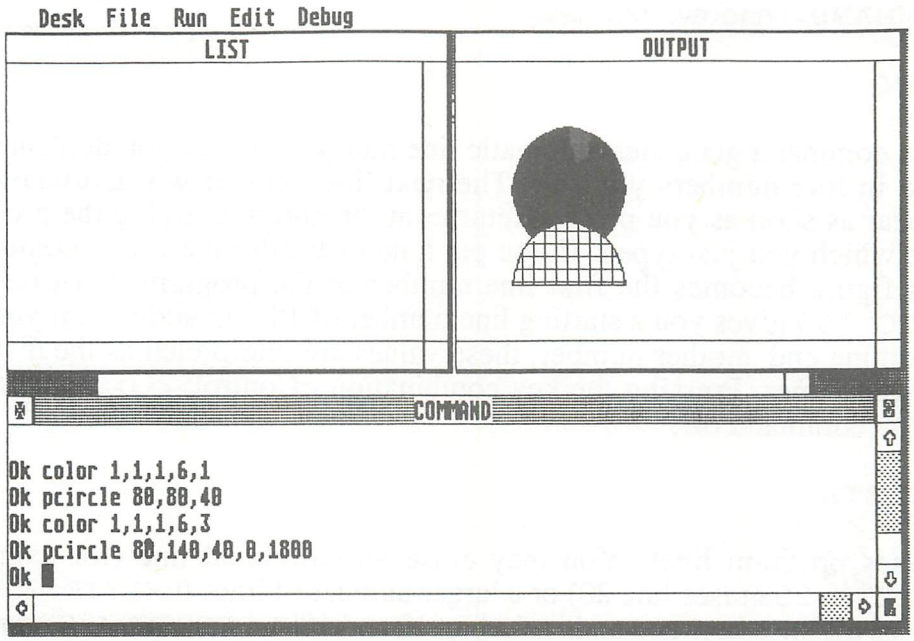
3.6.7 LINEF A,B,C,D

Draws a line in the **OUTPUT** window from coordinates A,B to C,D. A is the X coordinate of the starting point of the line in pixels and B is the Y coordinate of the starting point of the line in pixels. C is the X coordinate of the endpoint of the line in pixels and D is the Y coordinate of the endpoint of the line in pixels.



3.6.8 PCIRCLE A,B,C,D,E

Draws a circle in the **OUTPUT** window. The A and B indicate the X and Y coordinates, C determines the radius of the circle and D and E indicate the beginning and the end angles (in tenths of a degree) of the circle. Circles and circle sections are filled in with the COLOR command (COLOR *text color, fill color, line color, index, style*).



3.6.9 PELLIPSE A,B,C,D,E,F

This command draws an ellipse in the **OUTPUT** window, where A and B mark the X / Y coordinate of the X radius, C and D determine the Y radius, and the E and F indicate the beginning and the end angles (in tenths of a degree) of the ellipse. Both the ellipse and ellipse section are filled in with the predetermined COLOR (see 3.6.8.).

3.7 Useful BASIC tips

3.7.1 Programmer's aids

The following commands can be valuable to you when you're programming in BASIC. They are entered in direct mode (without line numbers) in the **COMMAND** window.

AUTO

This command generates automatic line numbering, i.e., you don't have to type in line numbers yourself. The next line number will automatically appear as soon as you press <Return> at the end of entering the program line which you just typed. If you put a number after the `AUTO` command, this figure becomes the first line number of the program. For example, `AUTO 100` gives you a starting line number of 100. In addition, if you add a comma and another number, these values are interpreted as the distance between lines. Pressing the key combination <Control><G> switches the `AUTO` command off.

DELETE

Erases program lines. You may erase an individual line (for example, `DELETE 20` erases line 20) or a larger number of lines (`DELETE 10-100` erases all lines between and including 10 and 100). `DELETE -20` erases all program lines from the beginning of the program up to line 20.

LIST

Lists a program. By adding a line number to the `LIST` command, you can display a specific line on the screen. For example, `LIST 10` displays only line 10. `LIST 10-100` displays lines 10 to 100, inclusive.

RENUM

Renumbers lines in memory. This command can be expanded for three values (parameters). `RENUM 10,100,5` will renumber the program, beginning with the original line 10, change this line to line number 100, and then increment the line numbers by 5.

3.7.2 Debugging

Here's a quick look at ST BASIC's aids for *debugging* (fixing errors):

1. You can stop a running program by pressing <Control><G>. The program may be continued by typing `CONT` and pressing <Return> or <Enter>.

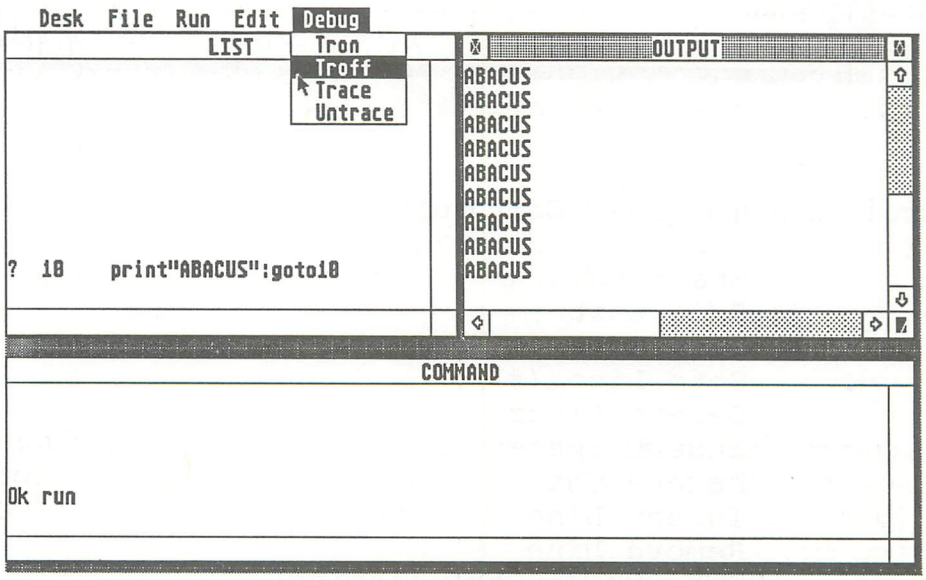
2. You can stop a running program by pressing <Control><C> (no continuation is possible).

3. `TRON` lets you trace the program line by line as it's running (indicates line numbers only). `TRON` is disabled by `TROFF`.

4. The `TRACE` command also allows you to trace the program line by line as it's running. Unlike `TRON`, `TRACE` displays the entire program line, including the line number. `TRACE` is disabled by `UNTRACE`.

5. `FOLLOW A` tells you the current value of the variable `A`.

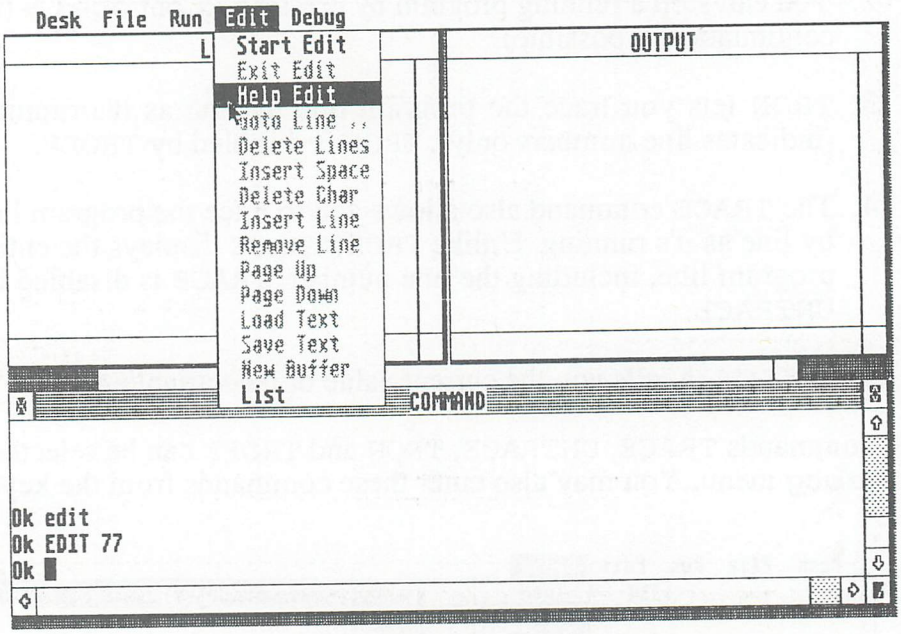
The commands `TRACE`, `UNTRACE`, `TRON` and `TROFF` can be selected from the **Debug** menu. You may also enter these commands from the keyboard.



3.7.3 The EDIT window

The **EDIT** window lets you freely revise and correct your program. You call the **EDIT** window and Edit mode either by typing EDIT in the **COMMAND** window, or by clicking Start Edit in the **Edit** menu.

For example, typing EDIT 20 displays line 20 in the **EDIT** window:

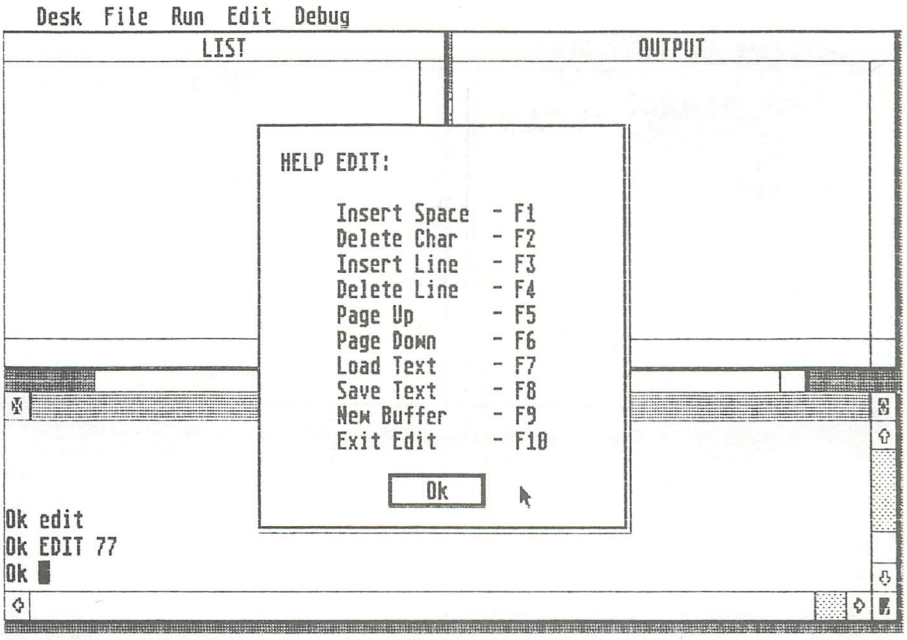


Here is the command set in **Edit** mode:

```
Start Edit
Exit Edit
Help Edit
Goto Line...(#)
Delete Lines
Insert Space
Delete Char
Insert Line
Remove Line
Page Up (in the listing)
```

```
Page Down (in the listing)
Load Text
Save Text
New Buffer
List
```

Furthermore, you can do some editing with the function keys. Clicking the option `Help Edit` in the **Edit** menu displays the function key layout.

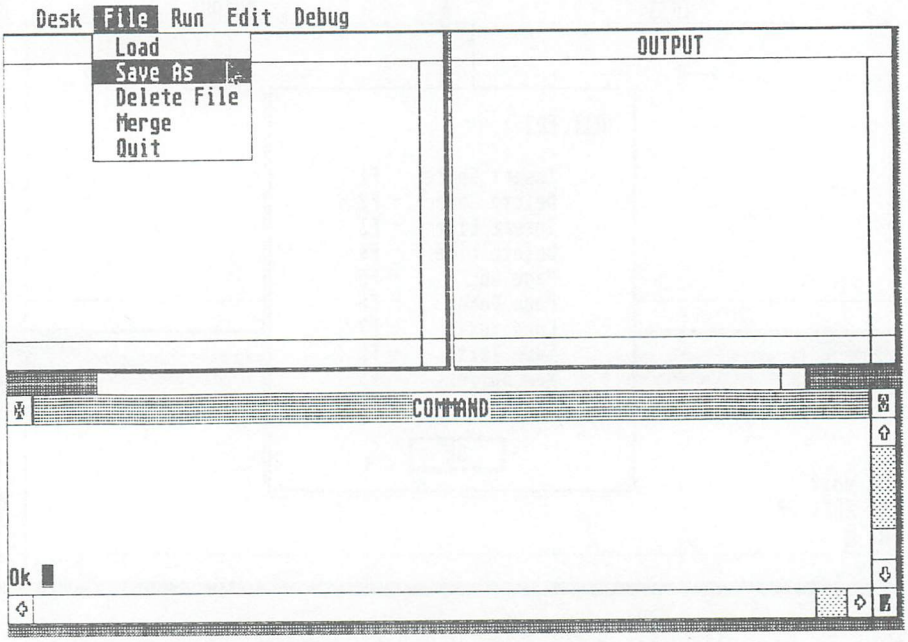


Editing is very simple. After selecting the **EDIT** window, you find the program listed there. Now you can correct the program line by line. The edited text is always listed in grey. Before confirming the changed line with the <Return> key, you can look at your work, store it (`Save text`) and also process it using the function keys listed above.

Now try writing a small program and experiment with editing a little. You'll see how easy it is to write programs on the ST.

3.8 Working with the disk drive in BASIC

Before you use a blank diskette, you must format it using the `Format...` program in the Desktop. You can save programming languages and your own programs on formatted diskettes. The drop-down menu **`File`** has everything you need. Have you entered a program that you want to use again? Then let's open the **`File`** menu:

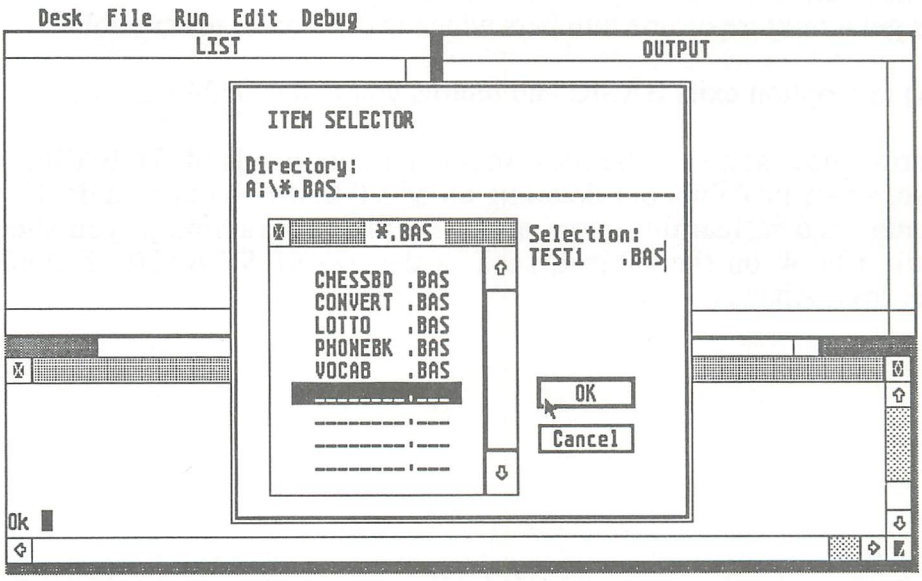


3.8.1 `Load` and `Save As`

Do you have a formatted diskette ready? Insert it in the drive and then click `Save As`. Now enter a name for our program. For example, type `TEST1`; the name appears below the word `Selection:` in the *file selector* dialog box. Click `OK`. Your program is now stored on the diskette for later retrieval.

Now we'll load the file we just saved. Your program name can be seen to the left in the item selector dialog box.

Select your program with the mouse, and click it. The filename appears in the `Selection:` field. When you click the `OK` button, the program is loaded into memory.



3.8.2 Delete File, Merge and Quit

We can erase a previously saved program from diskette with `Delete File`. This function is equivalent to moving a file icon to the trash can. If you select the `Delete File` option, the item selector dialog box is again displayed. Click the file you want to erase. The filename then is displayed below `Selection:`. It is deleted after you click `OK`.

Note: Be careful with this command—the `Delete File` option erases the file permanently!!

`Merge` lets you combine a program on diskette with another program already in memory. **Again, be careful with this command!!** Once the merge is performed, the two original programs are merged in memory, and only one remains—the merged program.

`Merge` is very useful for combining small program sections to create one large program. But be sure to renumber the programs to fit consecutively before you perform this command, or else things can get confusing. For example, Program 1 contains the line numbers from 10 to 150. You should renumber the second program so that it begins with line 160. Otherwise you'll get a mixture of line numbers which is difficult to unscramble.

The `Quit` option exits BASIC and returns you to the GEM Desktop.

We now know some of the most important commands of ST BASIC. Of course, space prohibits our detailing all of ST BASIC's commands. If you are interested in learning more about BASIC programming, you should consult a book on the subject, such as the *ATARI ST BASIC Training Guide* from Abacus.

3.9 First programs

Beginning computer users often try to deal with long program listings without any previous knowledge of BASIC. It's not surprising that finding errors is almost impossible, since no one is immune from making typographical errors. And computers do not overlook even the slightest typographical error.

You'll find BASIC programs on the following pages which contain the whole vocabulary introduced in this chapter. A detailed explanation of the program lines follows each program listing.

In some places you'll find commands that you haven't yet seen in this book. In these cases, the explanatory text should clarify things.

Now two hints:

- When you enter the following programs into your ST, don't forget to press the <Return> key at the end of each line.
- You can skip REM lines in the program explanations, since they are simply there for your reference.

3.9.1 Telephone book

This program lets you use your ST as a telephone directory. First enter the number of names you want in your directory, then the names and telephone numbers. When you run the program, simply enter a name into the ST. The program will find you the correct telephone number for that person.

```
10  REM Telephone book
20  CLEARW 2
30  INPUT "How many telephone numbers";A
40  REM dimensioning for numbers and names
50  DIM NA$(A), NU$ (A)
60  CLEARW 2
70  REM telephone listing
80  FOR N=1 TO A
90  INPUT "Name"; NA$ (N)
```

```
100 INPUT "Number"; NU$ (N)
110 PRINT
120 NEXT N
130 REM Search for Telephone numbers
140 CLEARW 2
150 INPUT "Which number are you looking for";A$
160 FOR N=1 TO A
170 REM Found name; print the number
180 IF A$=NA$ (N) Then GOSUB 210
190 NEXT N
200 GOTO 140
210 REM subroutine: Print the number
220 PRINT "The telephone number: "; NU$ (N)
230 PRINT
240 INPUT "Please press <Return>";B$
250 RETURN
```

Now let's look at the program line by line:

Line 20: Erases the **OUTPUT** window.

Line 30: Variable A stores the quantity of names and telephone numbers.

Line 50: Here the variables for the telephone numbers (numerical variable NU$(A)) and the names (string variable NA$(A)) are set. Although the telephone number is a numerical value, we had to select a string variable due to the slash (i.e., 616/241-5510).

Line 60: Erases the **OUTPUT** window.

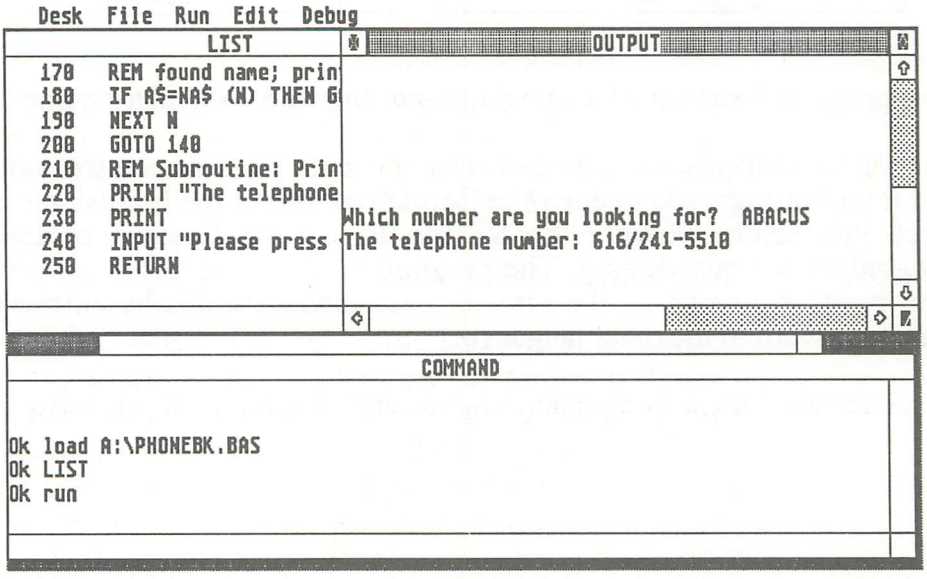
Line 70-120: Prompt for name (line 90) and his/her telephone number (line 100). All names are entered here according to your entry in variable A in line 30. Therefore, the shape of the FOR...NEXT loop for frequent runs of the same function are chosen (with differing variable contents). The prompting repeats until the value of variable A is reached.

Line 130-150: The program asks for the name for the person for whom you need a telephone number. This name is stored in variable A$.

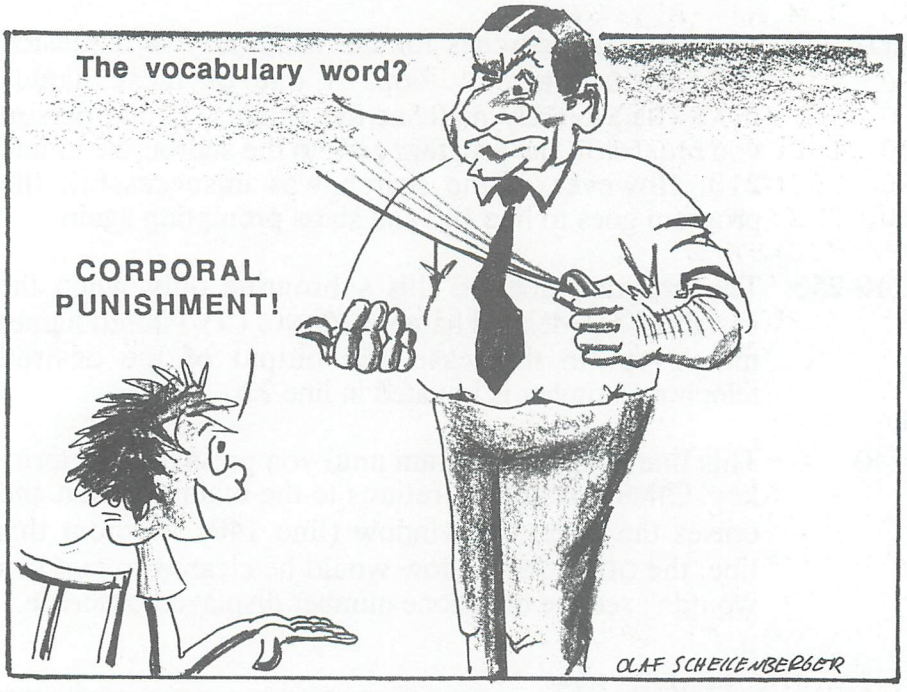
Line 160-190: The program searches for the name(s) you requested with a FOR...NEXT loop. If one of these names (NA$(N)) exactly matches the name(s) of the person you requested, the program goes to the subroutine in line 210. However, if the search was unsuccessful, the program goes to line 140 and starts prompting again.

Line 210-250: The program goes to this subroutine only when the variables A$ (desired name) and NA$(N) (stored name) match up. In this case, the output of the desired telephone number is initiated in line 220.

Line 240: This line halts the program until you press the <Return> key. Otherwise the ST returns to the main program and erases the **OUTPUT** window (line 140). Without this line, the **OUTPUT** window would be cleared so fast you wouldn't see the telephone number displayed onscreen.



3.9.2 Vocabulary program



This program turns your ST into a foreign language vocabulary trainer.

The ST will ask you for the number of vocabulary words you want stored, and it will prompt you to enter the German as well as the English meaning. Then you select whether the ST should use the German or English vocabulary for questioning. The program randomly searches for a word, then prints the word on the screen. You respond to this by entering the matching word in the other language.

You can substitute a foreign language other than German if you wish.

```
10  REM Vocabulary program
20  CLEARW 2
30  INPUT "How many words"; A
40  DIM A$ (A), B$ (A)
50  CLEARW 2
60  FOR N=1 TO A
70  INPUT "GERMAN MEANING";A$(N)
80  INPUT "ENGLISH MEANING";B$(N)
90  PRINT
100 NEXT N
110 CLEARW 2
120 B$=""
130 INPUT "GERMAN - ENGLISH (yes) "; B$
140 IF B$="yes" THEN GOSUB 200 ELSE GOSUB 280
150 PRINT
160 B$=""
170 INPUT "REPEAT (no)"; B$
180 IF B$ = "no" THEN END
190 GOTO 110
200 B=RND *A+1
210 CLEARW 2
220 PRINT A$ (B);
230 AN$=""
240 INPUT AN$
250 IF AN$=B$ (B) THEN PRINT "Correct!"
260 IF AN$<>B$ (B) THEN PRINT "Incorrect!"
270 RETURN
280 B=RND * A+1
290 CLEARW 2
300 PRINT B$ (B);
310 AN$=""
320 INPUT AN$
330 IF AN$ = A$ (B) THEN PRINT "Correct!"
340 IF AN$ <> (B) THEN PRINT "Incorrect!"
350 RETURN
```

Now for the line-by-line description:

Line 20: Erases the **OUTPUT** window

Line 30: The program asks how many words you would like to use and stores them and their translations in variable A.

Line 40: Here the number of the string variables in German (A$(A)) and in English (B$(A)) is determined.

Line 50: Clears the **OUTPUT** window.

Line 60-100: Prompt for the German meaning (line 70) and the English meaning (line 80). You can use a foreign language other than German. The number of words is determined by your entry for variable A in line 30. Therefore, the number of cycles of the FOR...NEXT loop are chosen. The prompting for data takes place until the value of variable A is reached.

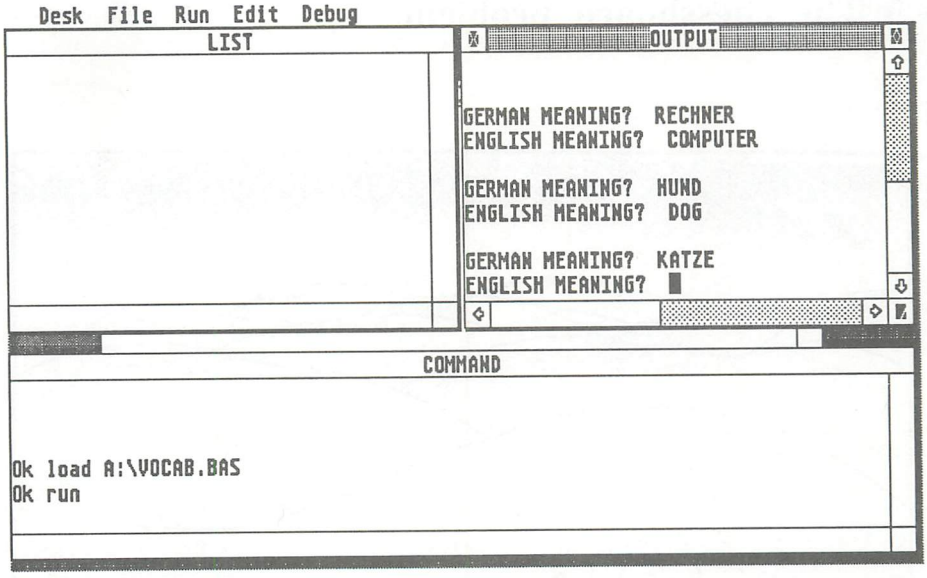
Line 110-130: After the **OUTPUT** window is cleared, you are asked whether you want to answer in German or English, and whether you want to be questioned in German or English. Your answer is stored in the string variable B$.

Line 140: This line determines to which subroutine your ST has to go. If you answered yes in line 130, then the ST jumps to the subroutine in line 190 (German-English); otherwise, the ST jumps to the subroutine in line 260 (English-German).

Here again is the command IF...THEN...ELSE. Line 140 can be translated into English as follows: If B$ = yes, then go to the subroutine in line 190—otherwise, go to the subroutine in line 260.

There are always two possibilities for our IF...THEN questioning. When the IF statement is correct, the command immediately behind the word IF is executed. If this is not the case, the THEN command is carried out behind the word ELSE. If ELSE is missing in the IF command, and the statement is not true, and the program continues with the next program line.

Line 150-190: After checking the vocabulary for clarity, a blank line is printed on the screen (line 150) The program then asks whether you want to continue with more words (line 170). The entry determines whether the program ends (with the END in line 180), or continue (line 190).



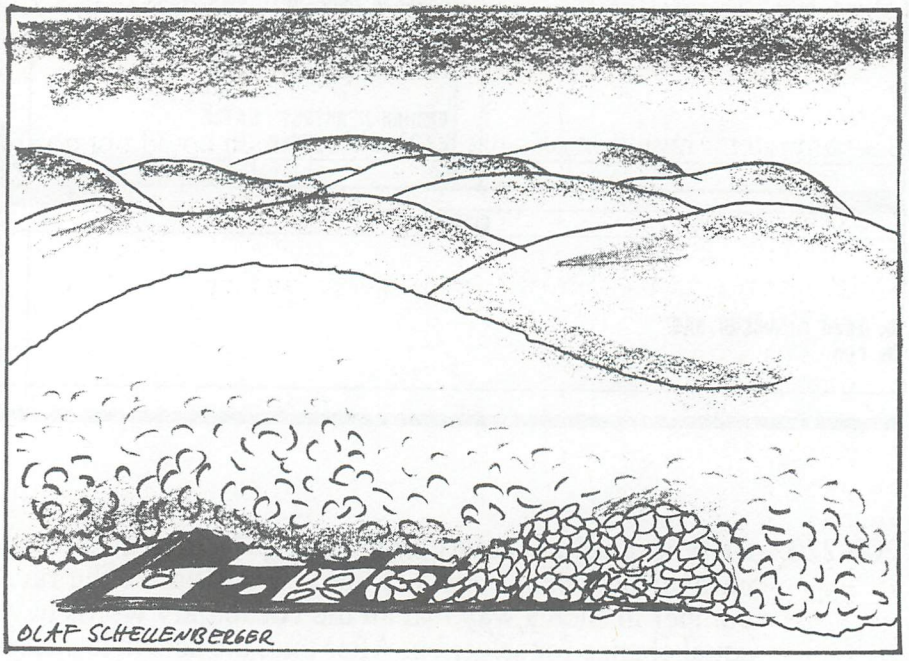
Line 200-220: `RND` is another new command. `RND` creates a random number between 0 and 1. Line 200 multiplies the random number in such a way that all the vocabulary words have an equal chance for selection. After that, the screen is cleared in line 210, and the word is randomly selected and placed on the screen in line 220. The semicolon after the `PRINT` command sets your answer directly after the definition.

Line 230-270: The program checks for a match between variables `AN$` and `B$(B)`. If they match, the answer was correct. An incorrect answer returns the program to its main section (line 270).

Line 280-300: The random number is multiplied with the formula in line 280 and incremented by 1, so that all words contained have an equal chance of being used. The screen is cleared in line 290, and the random word displayed on the screen (line 300). Again, the semicolon behind the `PRINT` statement places the answer to the question directly after the word .

Line 310-350: The program checks to see if variables `AN$` and `A$(B)` match. If they do, the question was answered correctly. If the answer was incorrect, the program goes to line 350, then returns to the main program.

3.9.3 The chessboard problem



Our third program shows how the ST can solve a large mathematical problem within a few seconds. The chessboard problem is based on the following story:

Once upon a time, a King granted a Wise Man his choice of reward, in gratitude for rescuing the pensive princess' Virtue from almost certain Temptation. As his reward, this Wise Man asked only for a normal chessboard with kernels of wheat arranged on the chessboard in a special manner. The kernels were to be placed on the 64 squares so that the first square had 1 kernel, the second had two kernels, the third square had 4 kernels, the fourth square had 8 kernels, and so on.

At first, the King laughed at the request, since it seemed to be easily fulfilled. Since there were so many kernels of wheat in a bushel, what would one more or less matter?

But when the King ordered his men to fulfill the wish, he saw how many kernels would be required to fill the chessboard. He then had much less concern for his daughter's sacred Virtue, because he saw that not even his entire kingdom could supply such a large amount of wheat. The Wise Man had asked for the impossible–and much to his Chagrin, got the King's daughter instead.

The Ende

Here's a computer simulation of what all the king's men could not do:

```
10  REM Chessboard problem
20  CLEARW 2
30  REM the chessboard has 64 squares
40  FOR N=1 TO 64
50  PRINT "square"; N
60  REM power formula
70  A=2^(N-1)
80  PRINT A; "Wheat kernels"
90  REM all wheat stored in variable B
100 B=B+A
110 PRINT "Total"; B; "Wheat kernels"
120 PRINT
125 FOR I=1 TO 500: NEXT I
130 NEXT N
```

The number of wheat kernels required is too large to count in the end. As the number is so large in square 63, you can imagine how big the number would be in square 64. Here is the line by line explanation:

Line 20: Erases the **OUTPUT** window

Line 40: Here a FOR...NEXT loop is opened to compute 64 squares.

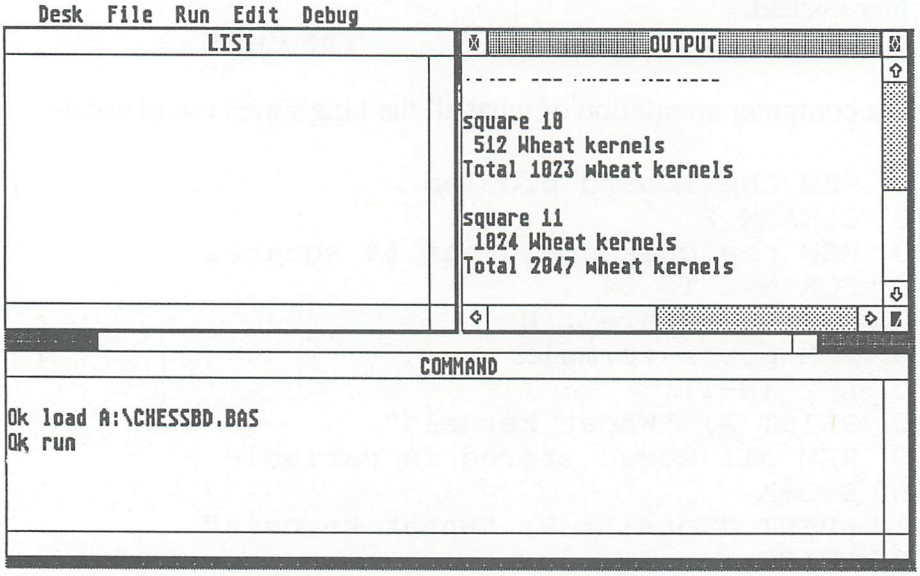
Line 50: This line outputs the square numbers on the screen.

Line 70: The number of the wheat kernels per square is calculated in agreement with this formula. The arrow pointing upward (^) is the exponential symbol. It's entered on the ST's keyboard by simultaneously pressing the <Shift> and <6> keys.

Line 80: The number of wheat kernels on the current square is displayed.

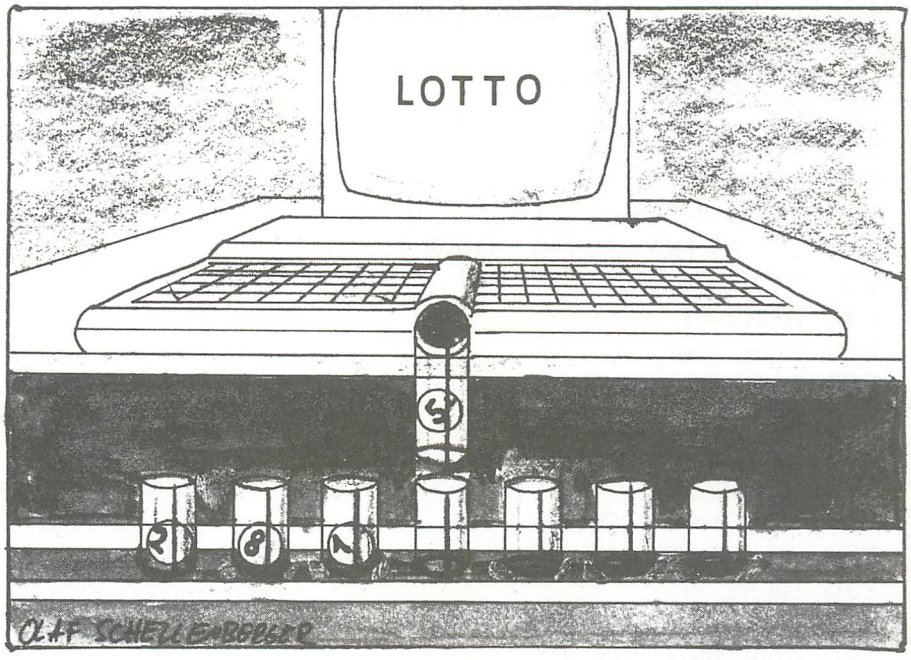
Line 100: The quantity of wheat kernels is stored in variable B and the value of the square already filled with wheat kernels is added.

Line 110: All wheat kernels on the chessboard are displayed.



Line 120-130: A blank line is placed on the **OUTPUT** window in line 120 for clarity, while line 130 jumps back to line 40 to calculate step by step all 64 squares on the chessboard. There is a pause in line 125 due to the blank FOR...NEXT loop, so that we may read the output on the screen.

3.9.4 Lotto numbers



This program shows you how your ST can help you win the lotto by giving you the correct winning numbers for the six-number as well as the seven-number lotto games. Like the foreign language vocabulary program, this program makes use of the RND command.

(Note: Abacus Software, Inc. makes no guarantees, written, verbal or implied, as to the accuracy of these lucky numbers. Program void where prohibited).

The program runs as follows: First you are asked whether you wish to play the six-number lotto (6 numbers between 1 and 49) or the seven-number lotto (7 numbers between 1 and 39).

Thereafter 6 or 7 numbers are drawn. These numbers are then carefully compared with each other before being placed on the screen, so that no more than two equal numbers are used. Then the lucky numbers are displayed on the screen.

```
10 REM Lotto numbers
20 CLEARW 2
30 INPUT"Six-number LOTTO (yes)";A$
40 IF A$<>"yes" THEN GOTO 180
50 FOR N=1 TO 6
60 A%(N)=RND*49+1
70 NEXT N
80 FOR N=1 TO 6
90 FOR M=1 TO 6
100 IF N=M THEN GOTO 120
110 IF A%(N)=A%(M) THEN GOTO 50
120 NEXT M
130 NEXT N
140 FOR N=1 TO 6
150 PRINT A%(N);
160 NEXT N
165 PRINT
170 GOTO 30
180 FOR N=1 TO 7
190 A%(N)=RND*39+1
200 NEXT N
210 FOR N=1 TO 7
220 FOR M=1 TO 7
230 IF N=M THEN GOTO 250
240 IF A%(N)=A%(M)THEN GOTO 180
250 NEXT M
260 NEXT N
270 FOR N=1 TO 7
280 PRINT A%(N);
290 NEXT N
295 PRINT
300 GOTO 30
```

Let's look at the program line by line:

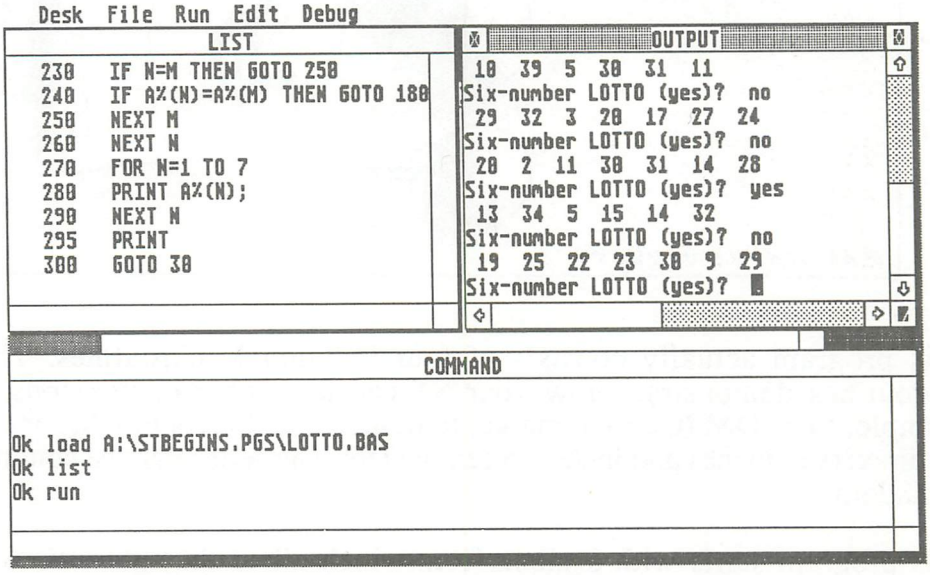
Line 10-30: After the screen is cleared (line 20), choose either the six-number lotto or seven-number lotto. Enter `yes` to play the six-number lotto. Enter something else in line 30 (i.e. `no`) to play the seven-number lotto. The program then moves on to the main program at line 180.

Line 50-70: These lines create the random numbers for the six-number lotto. The multiplication of `A%(N)` by 44 occurs because numbers between 1 and 44 are chosen in the lottery.

Line 80-130: All randomly selected numbers are compared in this line block. If two numbers are equal (line 110), new random numbers are produced by returning to line 50.

Line 140-170: The six different numbers are output. Good luck!

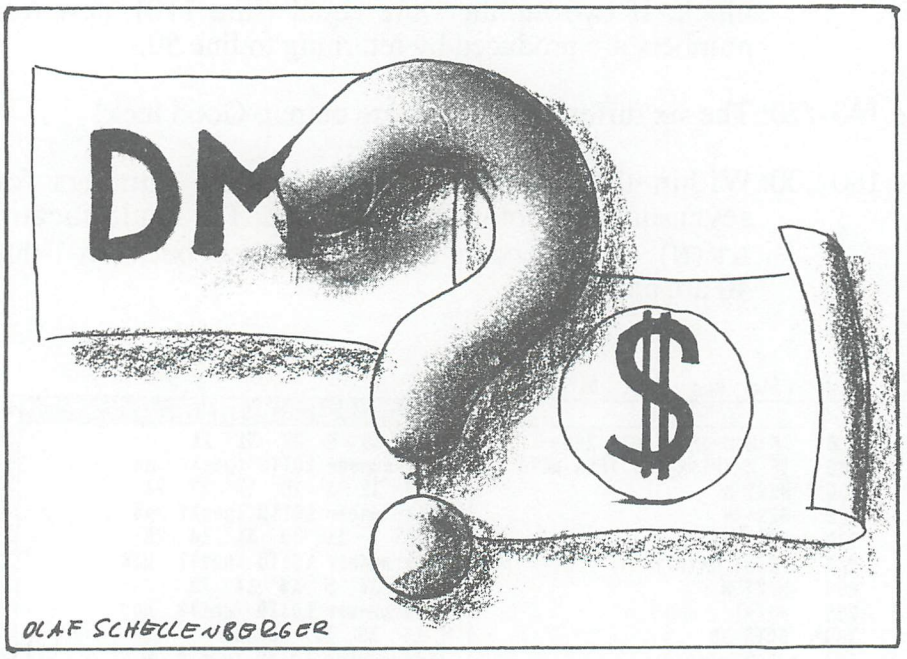
Line 180-200: Within these lines the seven random numbers for the seven-number lotto are produced. The multiplication of `A%(N)` by 40 occurs because numbers between 1 through 40 are chosen.



Line 210-260: In this line block, all randomly selected numbers are compared. If two numbers are equal (line 240) new random numbers are produced by returning to line 180.

Line 270-300: Here the seven numbers are output. They may be used for entry on the lotto ticket, including chance digit. Good luck!

3.9.5 Conversion program



This program actually consists of four individual subroutines. These subroutines demonstrate how your ST can remember conversions, for example, from DM (German marks) to dollars and dollars to DM; or from centimeters to inches and inches to centimeters. The series may be continued at random.

This program starts with a menu from which all of the subroutines are selected by pressing <Return>.

```
10 REM MENU
15 CLEARW 2
20 PRINT "PLEASE CHOOSE ONE:"
30 PRINT"1 CONVERT DM TO DOLLARS"
40 PRINT"2 CONVERT DOLLARS TO DM"
50 PRINT"3 CONVERT CM TO INCHES"
60 PRINT"4 CONVERT INCHES TO CM"
70 INPUT"WHICH CONVERSION WOULD YOU LIKE";X
75 REM Jump to appropriate subroutine
80 ON X GOTO 90,150,210,270
90 REM DM-dollar conversion routine
100 CLEARW 2
110 INPUT"PLEASE INPUT DM";A
120 B=A/2.67
130 PRINT A;" DM IS ";B;" DOLLARS"
140 END
150 REM dollar-DM conversion
160 CLEARW 2
170 INPUT"INPUT DOLLAR VALUE";A
180 B=A*2.67
190 PRINT A;"DOLLARS EQUAL";B;"DM."
200 END
210 REM cm-inch conversion
220 CLEARW 2
230 INPUT"Please input cm";A
240 B=A/2.54
250 PRINT"THAT'S";B;"IN."
260 END
270 REM inch-cm conversion
280 CLEARW 2
290 INPUT"PLEASE INPUT INCHES";A
300 B=A*2.54
310 PRINT"THAT'S";B;"CM"
320 END
```

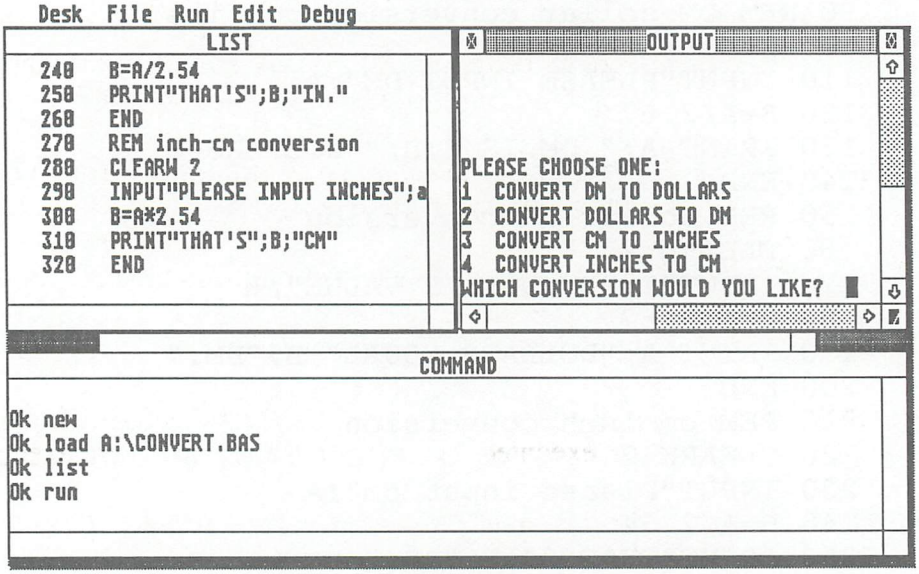
Line 10–70: This is the main menu. Figures from 1 to 4 are stored in X.

Line 80: This new command determines if X equals 1, 2, 3 or 4. Depending on the value it returns to the 1st, 2nd, 3rd or 4th line number behind GOTO.

Line 90–140: After the **OUTPUT** window is erased (line 100) the program asks for the DM amount desired (line 110). This

amount is divided by the actual dollar value—at the moment the dollar has the value of DM 2.67 (line 120). Next, the dollar value is placed in the **OUTPUT** window.

Line 150-200: After the **OUTPUT** window is erased the program asks for the dollar amount desired. This amount is multiplied with the actual dollar value (line 180). The DM amount is placed on the **OUTPUT** window (line 190).



Line 270-320: After the screen is erased the program asks for the inch value desired (line 290). This value is multiplied with the actual inch multiplier (line 300). In line 310 the centimeter amount is placed on the **OUTPUT** window.

3.10 Summary of fundamental ST BASIC commands

CLEARW *n*:	Clear screen windows— *n*=0 : **EDIT**; *n*=1 : **LIST**; *n*=2 : **OUTPUT**; *n*=3 : **COMMAND**.
CONT:	Program run continues after interruption by <Control><G> or the STOP command.
DIM:	Dimensions a set of variables (array).
END:	End of program. After this BASIC command the program cannot be resumed with CONT.
FOR...NEXT:	Start and end of a program loop, often used to increment a value by 1.
GOSUB...RETURN:	Jump to a subroutine then return to the main program.
GOTO:	Jump to a specific program line.
IF...THEN...ELSE:	IF a condition is true, the statements following THEN are executed, or statements after ELSE are executed.
INPUT:	Accept data into an existing variable; INPUT can also receive screen output.
LIST:	Display program lines on the LIST screen.
NEW:	Deletes program in memory.
PRINT:	Prints variables and texts on the screen.
REM:	Comments after this command serve for information only, and are ignored by the computer.
RND:	Produces a random number between 0 and 1.
RUN:	Start the program run.
STOP:	Program interruption—continue with CONT.

Chapter 4

ST LOGO

ST LOGO

4.1 A comparison of LOGO and BASIC

Now that we've learned the fundamentals of the BASIC language, we'll take a look at the second programming language included in the Atari ST package: ST LOGO. It differs from ST BASIC in the following respects:

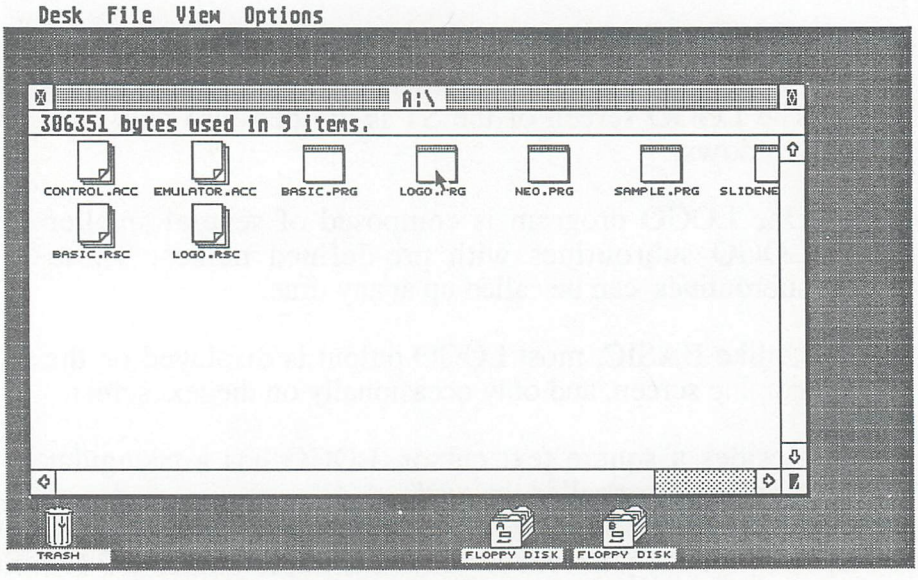
- LOGO programs do not use line numbers.
- The LOGO screen of the ST is divided into only two windows.
- One LOGO program is composed of several smaller LOGO subroutines with pre-defined names. These subroutines can be called up at any time.
- Unlike BASIC, most LOGO output is displayed on the graphic screen, and only occasionally on the text screen.
- Besides a square text cursor, LOGO has a triangular graphic cursor called the *turtle*.

This chapter will simply introduce you to the important LOGO commands, and show you something about the program structure of ST LOGO. For more information, you might want to refer to a more detailed book such as the *Atari ST LOGO User's Guide* published by Abacus.

4.2 Loading LOGO

After the computer has been switched on and the Desktop has appeared, list the Language Disk's directory by double-clicking the disk drive icon.

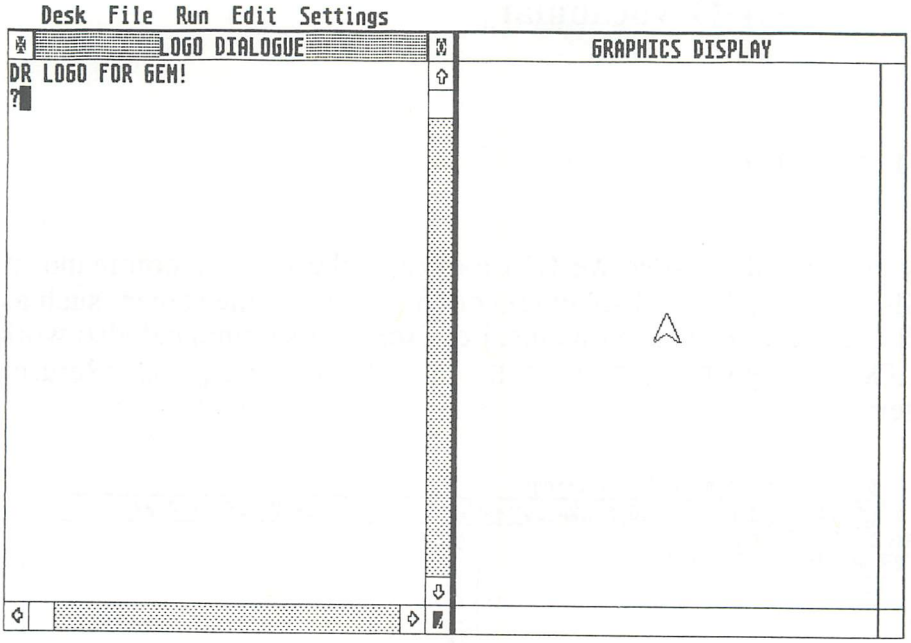
The Language Disk contains two LOGO programs. One is the program LOGO.PRG, the other a data file called LOGO.RSC:



There may be other files on your disk that are LOGO programs, marked by the file extension .LOG. We won't discuss these programs now. Instead, we intend to get acquainted with some basic commands in this chapter.

Load ST LOGO from your Language Disk by pointing the mouse pointer to LOGO.PRG and double-clicking that icon.

The mouse pointer changes into a bee, which signals that the disk drive is working. After a short wait LOGO.PRG is loaded into memory from disk, and now waits for your first command.



The screen on the left side is titled **LOGO DIALOGUE**. This is where your input (and some output) appears. The screen on the right side is called **GRAPHICS DISPLAY**.

As you can see, the menu bar is different and reads: **Desk**, **File**, **Run**, **Edit** and **Settings**. The turtle (the triangular object) is displayed at the center of the **GRAPHICS DISPLAY** screen.

The **LOGO DIALOGUE** screen greets us with the following message:

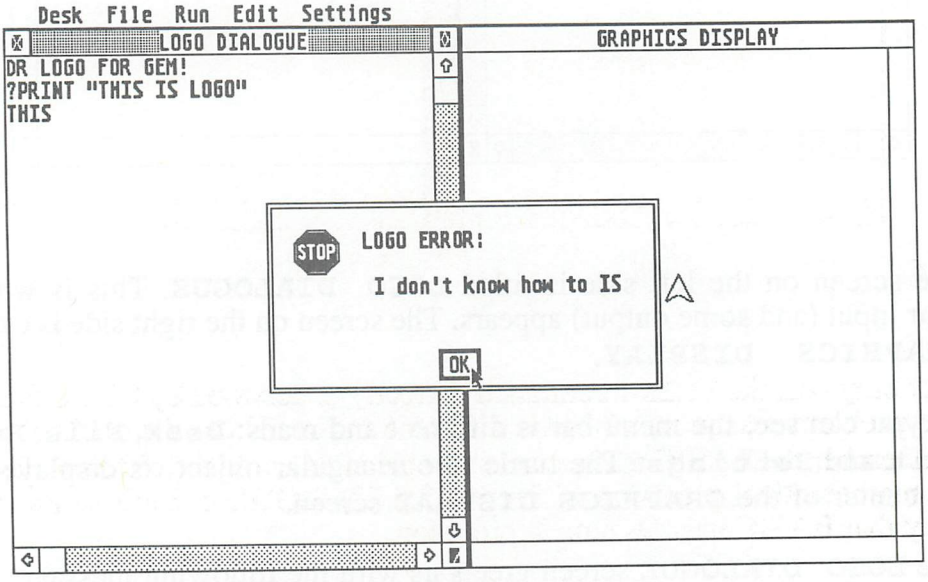
```
DR LOGO FOR GEM!
```

The "DR" term is a reference to the software firm Digital Research (DR), which developed both the GEM graphic operating system for your ST, and this ST version of the LOGO programming language.

4.3 The LOGO vocabulary

4.3.1 PRINT

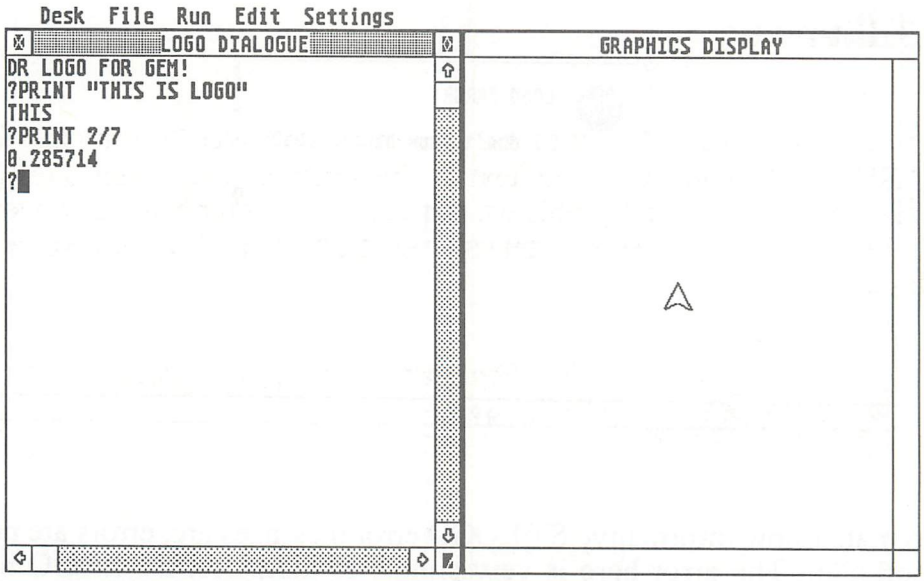
Do you remember when we first looked at the PRINT command in ST BASIC? PRINT let us display text or characters on the screen, such as the results of mathematical problems. Let's see if this command also works in ST LOGO. Type PRINT "THIS IS LOGO" and press <Return> or <Enter>.



The first noticeable difference from BASIC is that, without pressing the <Caps Lock> key, ST LOGO prints all lettering on the screen in upper case.

But what does the error message in the center of the screen mean? ST LOGO did not understand what was meant by the word "IS". In other words, in LOGO the PRINT command only affects the word immediately to the right of the quotation mark. The first word "THIS" appeared on the screen as a command entry.

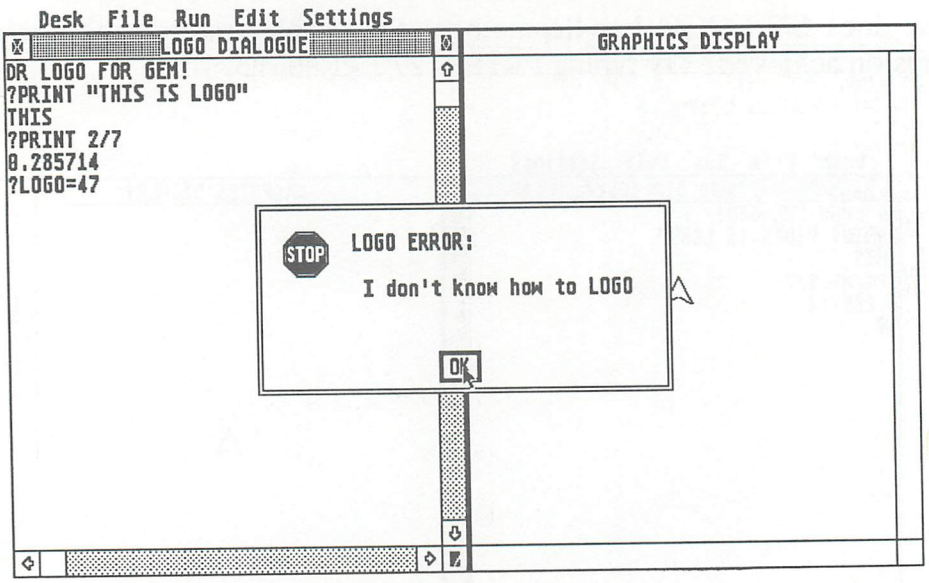
How does ST LOGO handle mathematical problems and the decimal precision achieved? Try typing `PRINT 2/7` <Return>.



Not only was the `PRINT` command correctly understood by ST LOGO, but the problem was precisely solved up to 6 digits behind the decimal point (remember the BASIC mathematical problem `2/7`, and the different degrees of mathematical precision in Section 3.5.2?). LOGO did this in the same way that BASIC handles simple precision.

4.3.2 MAKE

Is it also possible to store variable values in the computer's memory with ST LOGO? Let's try it as if we were still in BASIC—type in a variable declaration such as `LOGO=47`, press <Return> or <Enter> and watch what happens:



No matter how informative ST LOGO error messages are, errors are never a good sign. The error here is very simple to diagnose: ST LOGO has no such word as "LOGO" in its collection of language elements. ST LOGO searched in its vocabulary for the word LOGO, and after finding no such word, displayed an error message.

When you assign a value to a variable in ST LOGO, you must observe two rules:

1) The assignment must begin with the word MAKE.

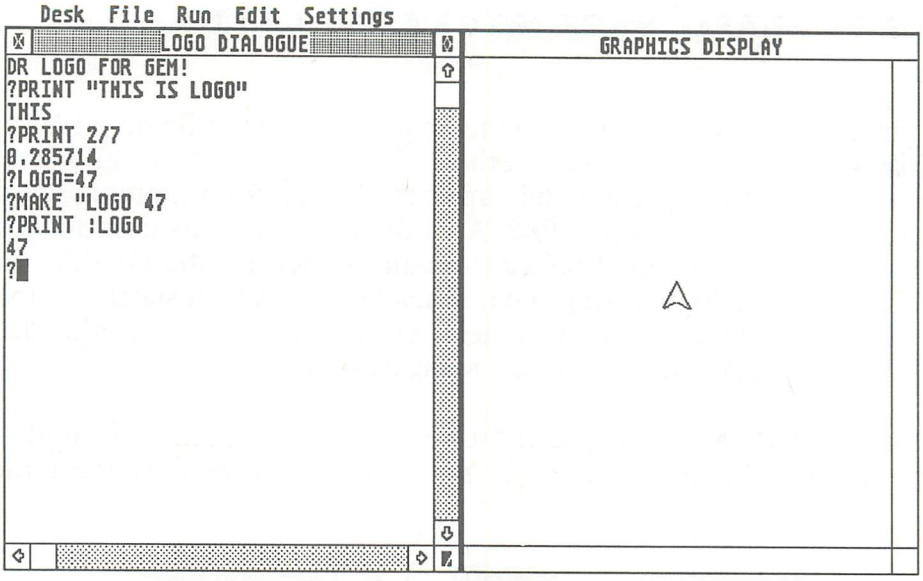
2) The equal sign (=) is not correct for the assignment. The variable assignment in LOGO is made with a single quotation mark and a space.

For example:

```
MAKE "LOGO 47
```

The variable can be viewed with PRINT : (don't forget the space and colon characters preceding the variable name):

```
PRINT :LOGO
```



4.3.3 CS and CT

CLEARW was the BASIC command to erase the different windows. ST LOGO also has commands that erase the two screens.

CS clears the graphic screen (**GRAPHICS DISPLAY**) and CT deletes the text screen (**LOGO DIALOGUE**).

Although the LOGO language excels in graphics display, it is not limited to graphics alone.

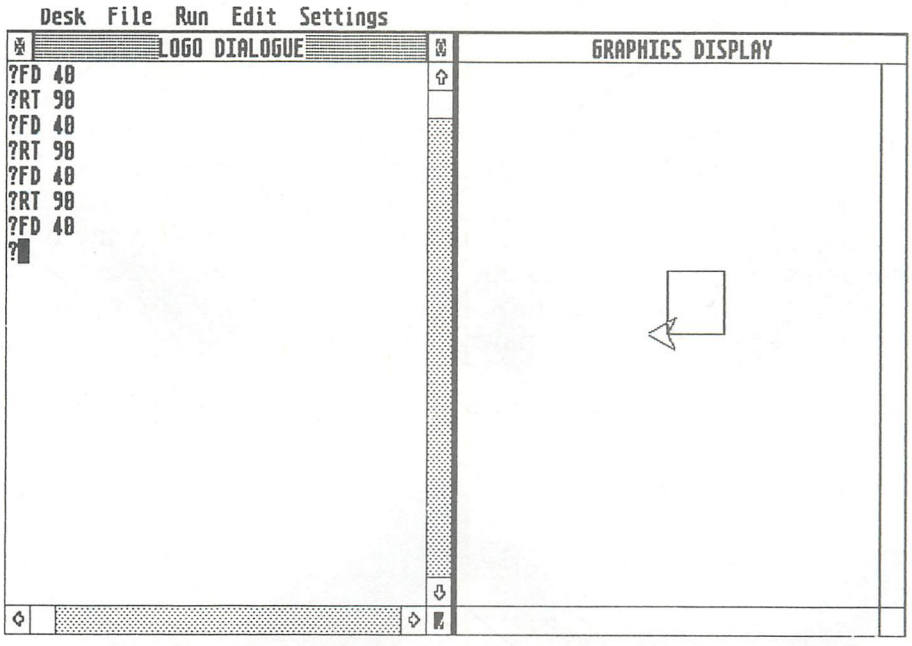
4.3.4 FORWARD, BACK, RIGHT and LEFT

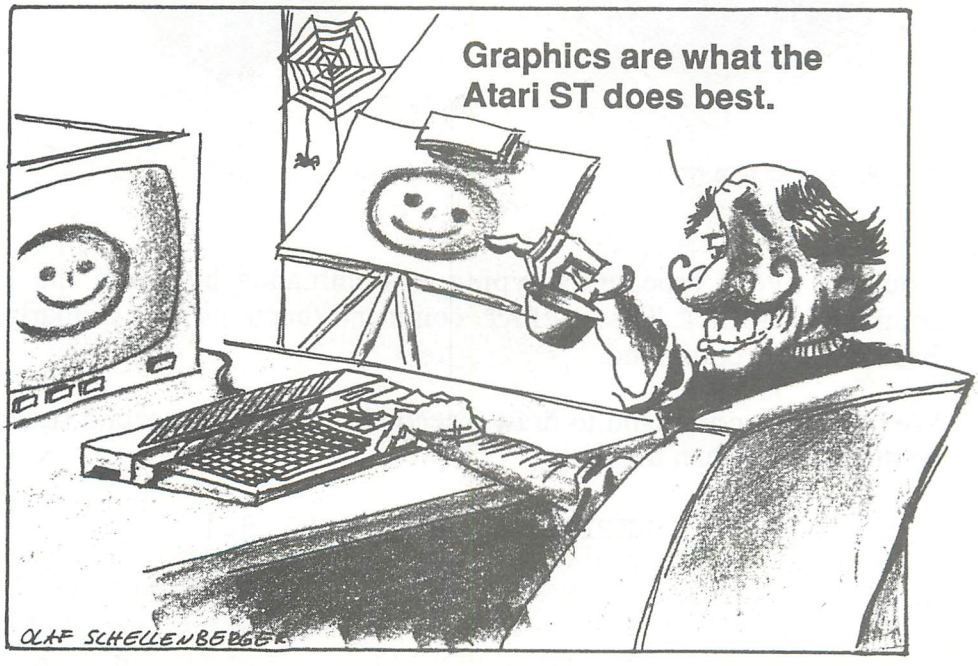
Originally LOGO was developed to acquaint young children with logical thinking through a simple method. Since children usually express themselves best with pencil and paper, the LOGO programming language works as graphically as possible. A cardboard turtle was used for logical operations. It had a pencil poked through its shell for drawing the turtle's movements. The logical steps could then be traced on a sheet of paper on the floor (move forward, turn left, etc.). These steps could also be performed repeatedly, like a spider spins its web.

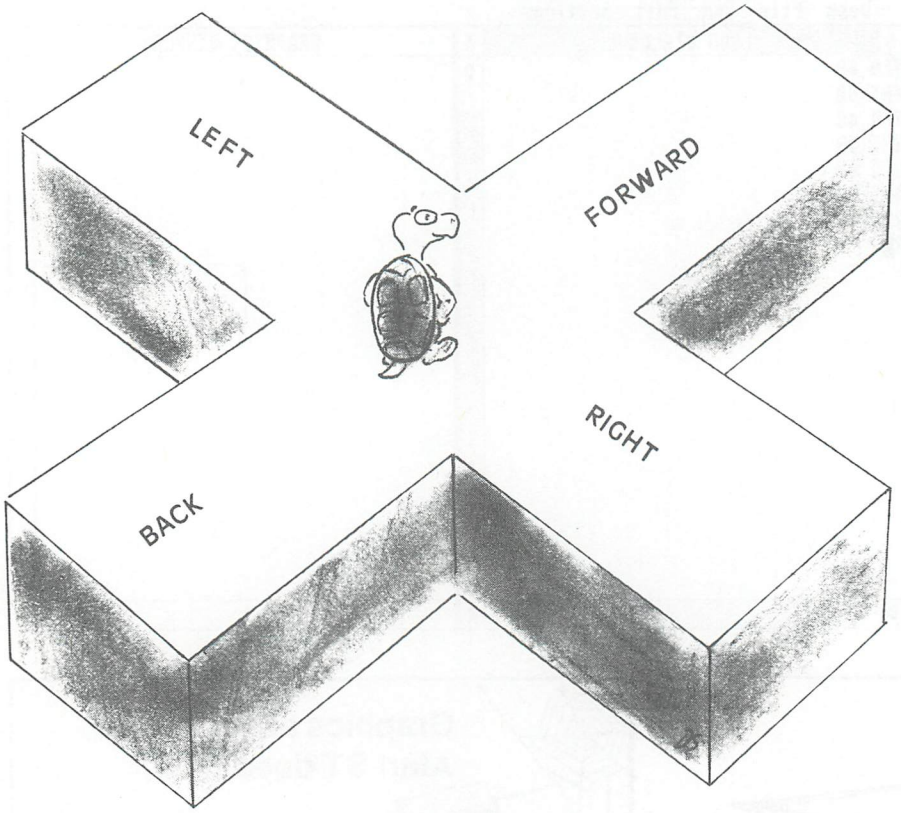
Look at the **GRAPHICS DISPLAY**. At the moment, a triangle (the turtle) is located in the center of the screen. How can we move this turtle?

We move the turtle with the LOGO commands FORWARD (abbreviation FD) and BACK (abbreviation BK). We have to indicate the number of points (distance) which your turtle is to move. The same is true if we want to move the turtle RIGHT (abbreviation RT) or LEFT (abbreviation LT). Use the following command sequence to draw a square:

```
FD 40 RT 90 FD 40 RT 90 FD 40 RT 90 FD 40
```


Desk File Run Edit Settings
LOGO DIALOGUE
?FD 40
?RT 90
?FD 40
?RT 90
?FD 40
?RT 90
?FD 40
?
GRAPHICS DISPLAY


Graphics are what the Atari ST does best.
OLAF SCHELLENBERGER

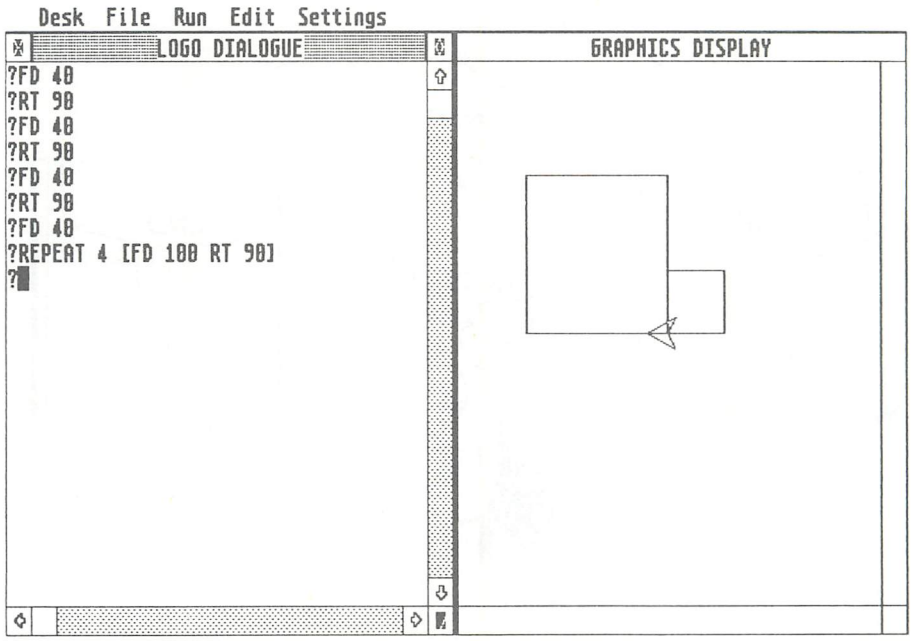


4.3.5 REPEAT

You can avoid repeatedly typing in commands by using the LOGO command REPEAT. The REPEAT command functions very similarly to the BASIC GOTO command.

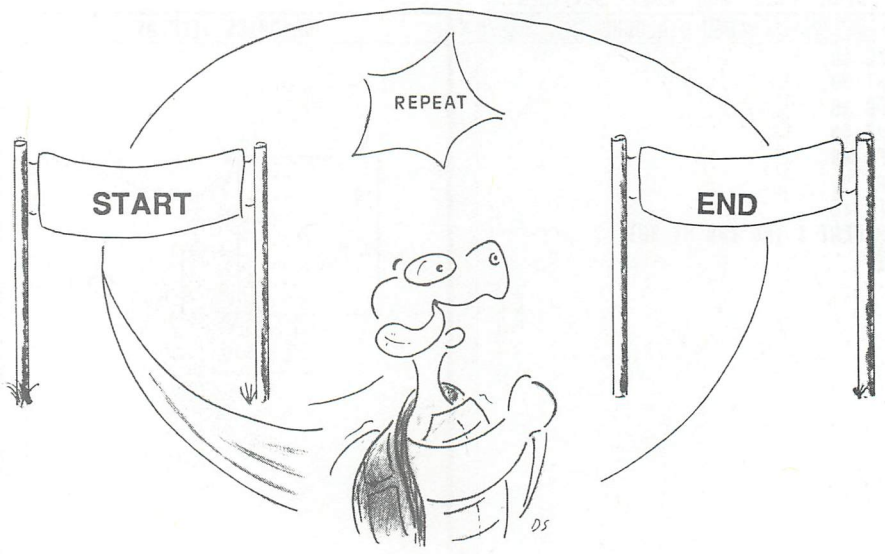
We'll use this command to draw a second square. This second square will be much larger than the first:

```
REPEAT 4 [FD 100 RT 90]
```



This command sequence is much easier than typing in seven individual commands. Don't forget to put the commands to be REPEATed in brackets following the REPEAT command itself. You could even put our pre-defined variable LOGO (=47) into the loop:

```
REPEAT 4 [FD :LOGO RT 90]
```

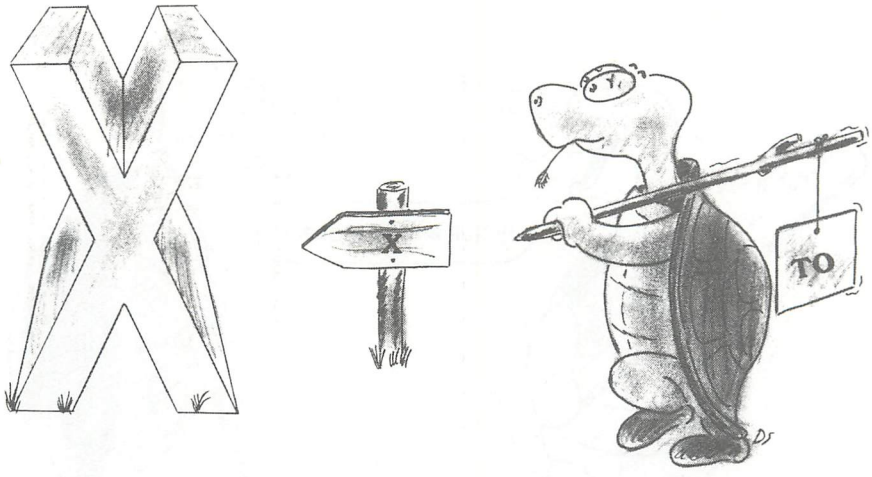


4.3.6 TO

The beginning of this chapter mentioned that one of the differences between ST LOGO and BASIC is that ST LOGO programs use several small subroutines instead of line numbers.

A *subroutine* in ST LOGO may be shorter than a REPEAT loop. You only have to start the subroutine entry with the word TO and close it with END. Between TO *subroutine name* and END, you may add as many program steps as you like.

The advantage of ST LOGO subroutines is that later on, individual routines may be directly called, or called from other subroutines by their names.



We'll construct two small subroutines:

```
TO TEXTCLEAR
CT
END

TO GRAPHICCLEAR
CS
END
```

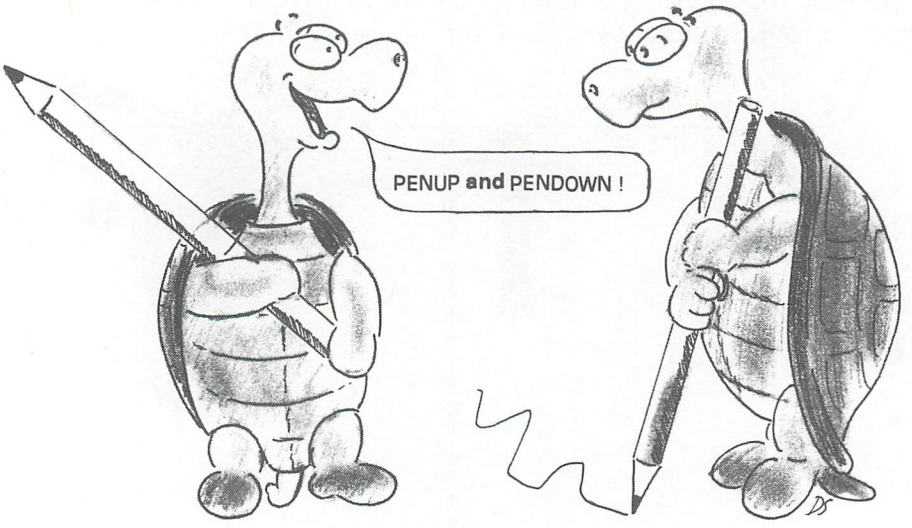
What happened as you typed these two LOGO subroutines, and what effect did these programs have?

After you defined the TEXTCLEAR subroutine, the question mark at the beginning of the entry line below changed to a greater than symbol (>). This symbol was displayed until you entered the END command.

The effect of these two LOGO subroutines that you can now use either the "real" LOGO command CS or the newly-defined command GRAPHICCLEAR. Both have the same result: the graphic screen is erased when you press <Return>.

This means that ST LOGO lets you create new commands for your own convenience, or use English words for LOGO vocabulary.

4.3.7 PENUP and PENDOWN

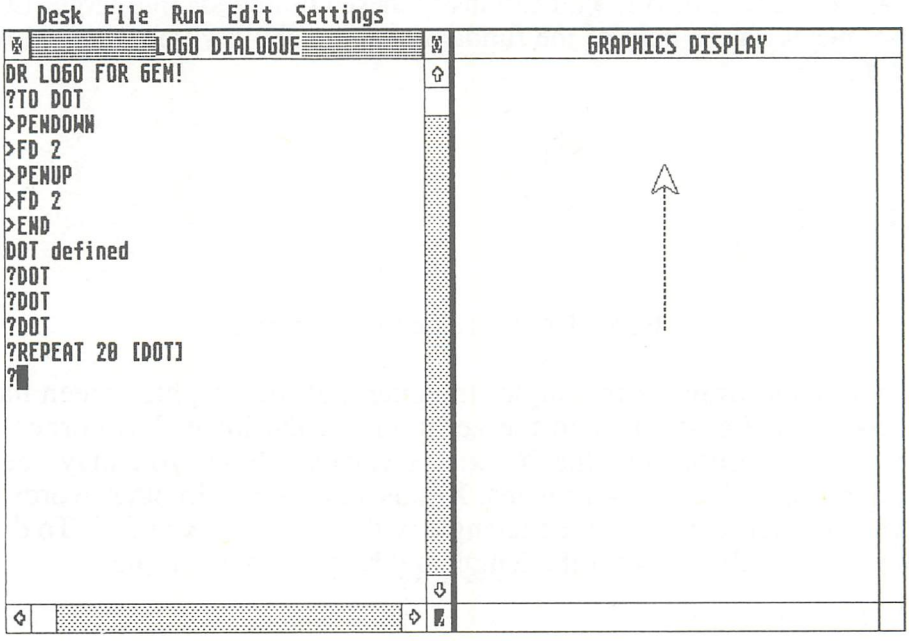


We talked briefly about the development of ST LOGO, when the kindergartners placed a pencil into their cardboard turtle to trace its path. But they could also remove this pencil from the turtle. LOGO gives us these same functions with the commands PENUP (do not draw) and PENDOWN (pen ready to draw). Both commands control the drawing capabilities of our turtle on the **GRAPHICS DISPLAY** screen.

Let's use the commands PENUP and PENDOWN in a short program that draws one point on the **GRAPHICS DISPLAY** screen, then moves the pen up so that the next point remains blank so that eventually a dotted line is produced. Let's call this subroutine DOT:

```
TO DOT
PENDOWN
FD 2
PENUP
FD 2
END
```

You have just defined a procedure with the name DOT. Start the program by repeatedly typing DOT<Return>. A dotted line appears on the screen. You can also use the REPEAT function to execute DOT, if you would prefer not to type the routine name over and over:



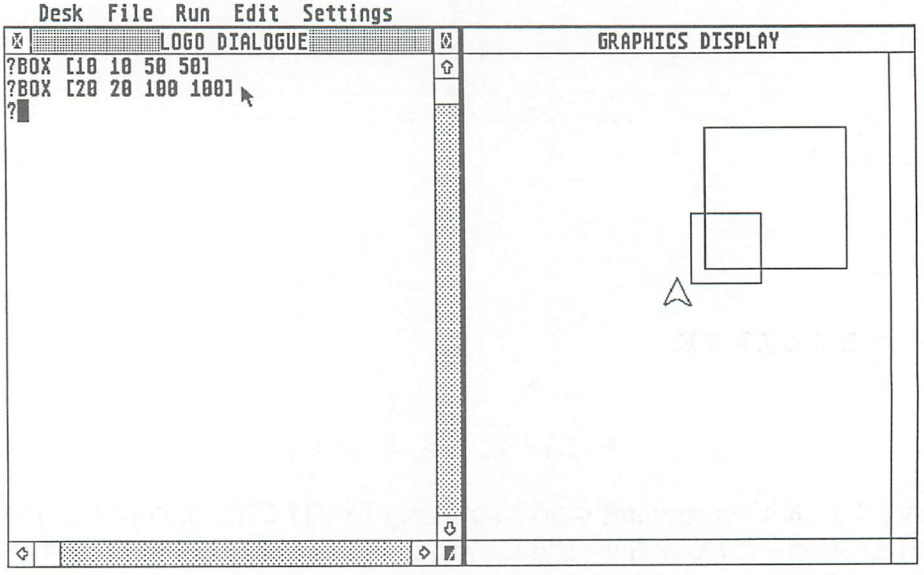
4.4 More LOGO commands

We'll introduce you to more ST LOGO commands in this section. The main purpose of these LOGO commands is to accomplish the greatest effects with the least amount of effort. You can insert these commands into larger LOGO subroutines with the help of the fundamental vocabulary you now know for ST LOGO.

4.4.1 BOX

```
BOX [X Y length height]
```

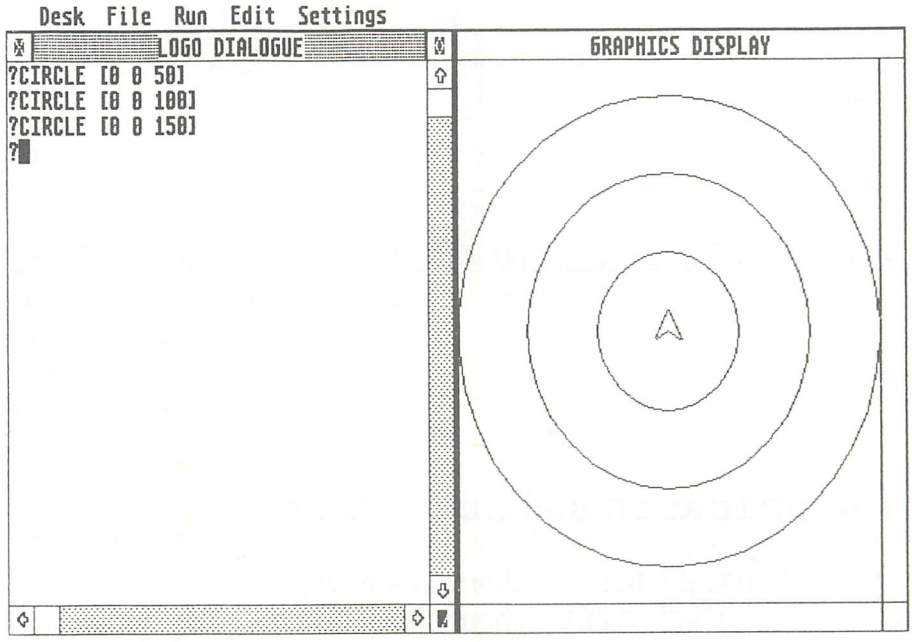
This command draws a rectangle. Imagine that the graphic screen has an X-Y coordinate system, with the zeropoint in the lower left corner. The X-axis is horizontal and the Y-axis is vertical. Now you may set any starting point with two coordinates: X-axis and Y-axis. In other words, you set the lower left corner of the rectangle with the values X and Y. To draw a square, use equal values for the length and height. For example:



4.4.2 CIRCLE

```
CIRCLE [A B C]
```

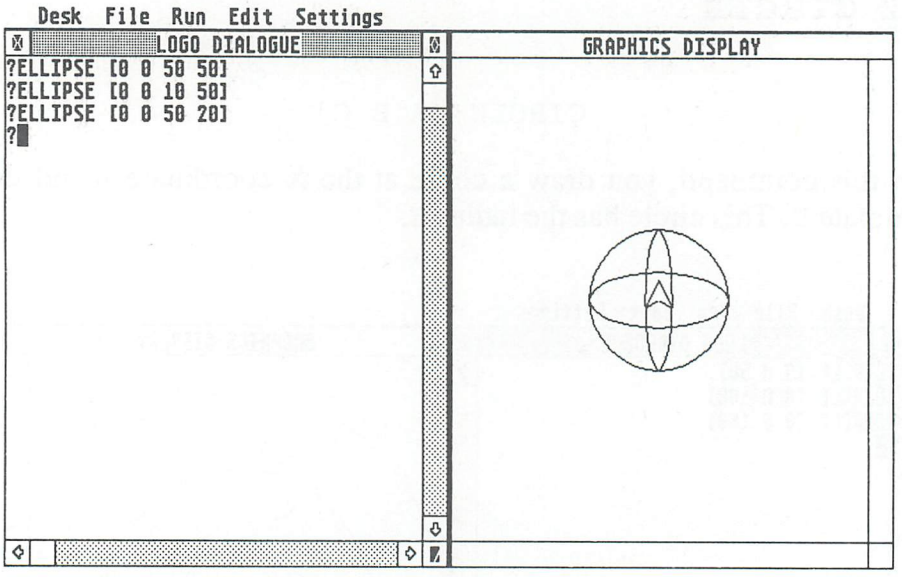
With this command, you draw a circle at the X coordinate A and the Y coordinate B. This circle has the radius C.



4.4.3 ELLIPSE

```
ELLIPSE [A B C D]
```

This ST LOGO command works similarly to CIRCLE, except that the two radii entered—C (X-radius) and D (Y-radius)—determine to what extent the ellipse will be drawn. If the C and D values are equal, a circle is drawn.



4.4.4 HIDETURTLE and SHOWTURTLE

HIDETURTLE (abbreviation HT)
SHOWTURTLE (abbreviation ST)

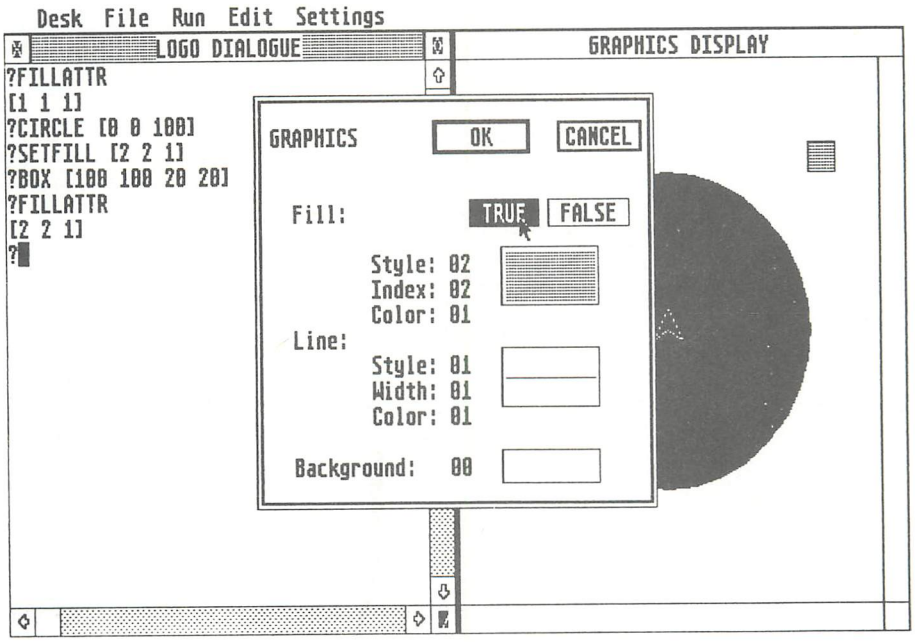
HIDETURTLE erases the turtle from the **GRAPHICS DISPLAY** window. It will reappear if you enter SHOWTURTLE. However, the drawing functions will remain even when the turtle is invisible.

4.4.5 FILLATTR and SETFILL

```
FILLATTR
SETFILL [A B C]
```

You may fill enclosed surfaces with a pattern. Choose Graphics from the **Settings** menu and click TRUE at the Fill option dialog box. Then you may choose one of the many patterns available in ST LOGO.

FILLATTR calls up the current fill color. SETFILL [A B C] lets you change parameters according to your own tastes. The A indicates the type of drawing style and can accept the values 0 to 4. The B indicates the index, and can accept values between 0 and 12. If you have a color monitor, you may choose up to 16 colors. The colors can be adjusted with the command SETFILL and the parameter C.



4.4.6 SHUFFLE and SORT

The SHUFFLE command randomly mixes the numbers in the square brackets following the command. SORT arranges the numbers in ascending sequence.

4.4.7 MOUSE, NODES and TURTLEFACTS

MOUSE informs you of the coordinates of the mouse pointer, and which of the two mouse buttons is being used. NODES indicates the available memory space (1 node=4 bytes). TURTLEFACTS informs you of the turtle's attributes.

4.5 The ST LOGO vocabulary—review

BACK	Move the turtle in the **GRAPHICS DISPLAY** window the indicated number of points backward.
CS	Clear **GRAPHICS DISPLAY** window.
CT	Clear **LOGO DIALOGUE** window.
END	End of an ST LOGO procedure introduced with TO.
FORWARD or FD	Move turtle in **GRAPHICS DISPLAY** window the specified number of points forward.
LEFT or LT	Rotate turtle in **GRAPHICS DISPLAY** the specified number of degrees left.
MAKE	Assign one value into a variable. For example, MAKE"A 55 gives A the value 55.
PENDOWN or PD	Drop turtle pen. Turtle now leaves a visible trail.
PENUP or PU	Raise pen on **GRAPHICS DISPLAY** turtle. The turtle can now be moved without drawing.
PRINT	Display variables and text in the **LOGO DIALOGUE**. If text in quotes after PRINT, only letters up to the first space are used. Variables must be preceded by a colon. Math problems are displayed onscreen without PRINT.
REPEAT	Repeat execution of the command sequence stated. For example: REPEAT 10 [FD 10] moves the turtle on the **GRAPHICS DISPLAY** forward 10 times, 10 points each time.
RIGHT or RT	Rotate turtle in the **GRAPHICS DISPLAY** window the specified number of degrees to the right.
TO	The name after TO is defined as a procedure, and has the same effect as a pre-defined ST LOGO command.

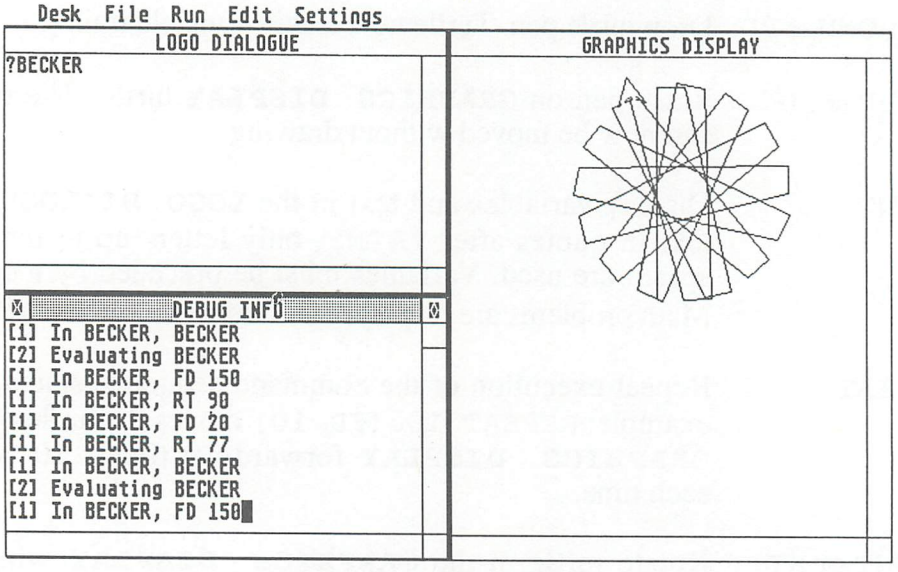
4.6 Debugging

1) Interrupt the running program by clicking the `Pause` function in the drop-down menu **Run**. Continue the program run by clicking `Continue` in the same menu.

2) Stop the running program by simultaneously pressing <Control> and <G> (the program cannot be continued).

3) Click `Watch` in the drop-down menu **Settings** (confirmed with a check mark)—the current procedure will be displayed during the program run in the `DEBUG INFO` window. It also indicates to which procedure the system jumps, and how many jumps have already taken place between the different procedures. Clicking `Watch` again switches this `DEBUG INFO` window off.

4) The `TRACE` command shows the procedure being carried out in `DEBUG INFO`. Click it off from the **Settings** menu again.



Chapter 5

A lesson in hiSTory

A lesson in hiSTory

Now that you are familiar with your ST, we would like to provide you with some background information. We'll try to explain some of the operations taking place inside a computer that you didn't understand from reading the previous chapters.

Furthermore, you might be interested to know how the ST was developed, and what else you might be able to do with this equipment in the future. We will discuss all of these topics and more in this chapter.

5.1 The Computer Age

5.1.1 The early days

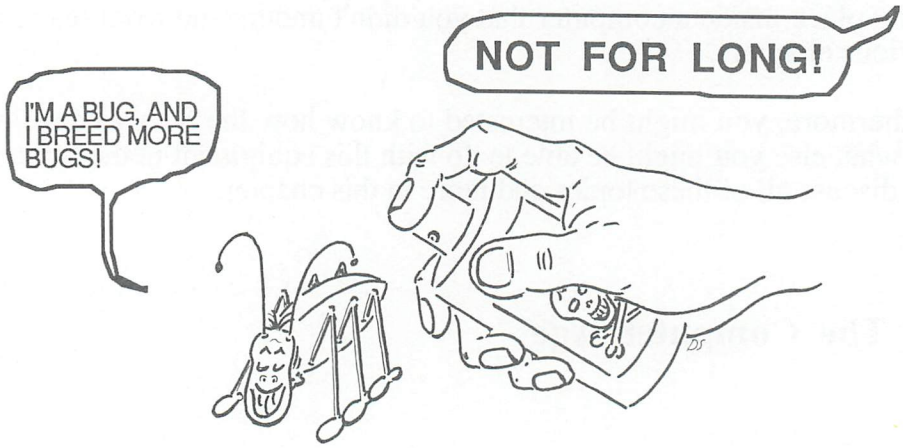
If you ever go to Munich, West Germany, visit the National Museum's computer history exhibit. In one corner of that display you'll find a huge calculator built in 1941 by the German Konrad Zuse. It was the first freely-programmable computer. It takes up a lot of space, and is extremely inefficient compared to modern computers. The low efficiency and enormous size of the Zuse computer are interrelated, since it was built from mechanical parts, rather than electronic components.

We'll take a brief look at how the Zuse computer operates. The Zuse computer worked with a complex array of electromechanical parts. The mechanical relays retained and conveyed information on the basis of which components were switched on, and which were off.

All the functions were handled by electromechanics in the Zuse computer. The machine made a great deal of noise due to clattering relays. There were numerous breakdowns with so many mechanical parts, and there were many more errors in calculation than with modern computers.

Also, the programs were full of errors, which caused frequent malfunctions and incorrect functions—and from those came incorrect results.

By the way, the expression "debugging" originates from this time period. It means to look for and get rid of errors in a computer program. Legend has it that the first bug found was an actual insect—a moth stuck in the relay of a 1940s American computer system.



Some years later, the invention of the transistor in 1948 by American engineers J. Bardeen, W. H. Brattain and W. B. Shockley dramatically altered the future of computers. The transistor is a much smaller electronic unit than a relay, performing the same work but working electronically instead of mechanically (on/off logic is one of its responsibilities). A transistorized computer was quickly developed, which was considerably smaller than previous electromechanical computers.

Inventors and developers continued work in this field. Finally, a process developed of combining a set of transistors into one small unit on a semiconductor using drawings of circuits. The drawings were done two-dimensionally—this was a definite step backward in the process of miniaturizing computer circuitry, and the first boards were probably a mess because of inaccuracies.

Then somebody had the good sense to use photographs of circuits instead of drawings. Now a drawing of several square meters could be reduced to a size of only a few square centimeters.

The final product made from this photographed group of transistors was created through a special chemical process. The integrated circuit, or *chip*, was born. One chip can perform tasks once assigned to many thousands of transistors. Pins emanate from the chip to connect it to the outside world—another circuit, perhaps another chip.

Compared to relays and transistors, the chip is much faster—up to 100,000 instructions per second may be executed. This is possible because chip connections send electricity over very short distances.

How far can integrated circuits develop? Well, one-megabyte chips, which will have one million transistor functions on a surface approximately the size of an aspirin tablet, are currently being mass-produced. The ST does not use these chips as of yet. However, a 520ST has sixteen 256-kilobyte chips that combine thousands of transistor functions onto surfaces about the size of a pinhead.

5.1.2 Cashing in on chips

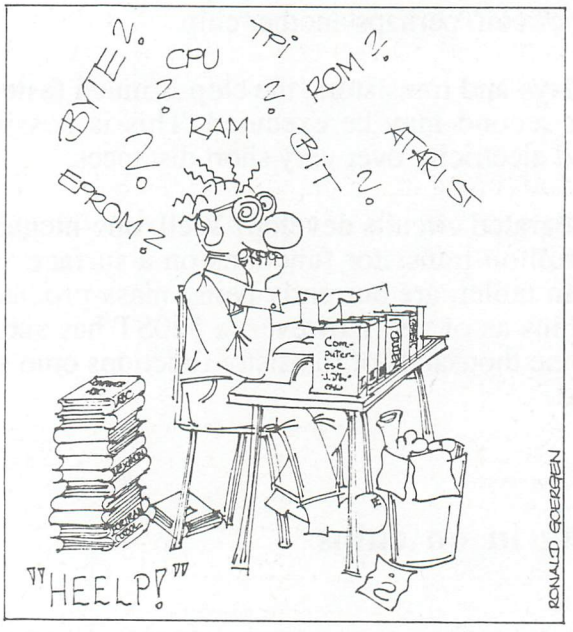
Chips can be used for many applications. Depending on their purpose, they may be purchased preprogrammed, or ready for your own programming.

The term *memory* was carried over to computer science to denote different storage capacities and abilities of chips. These different types of memories go under the following common general names:

ROM
PROM
EPROM
RAM

ROM is an abbreviation for Read Only Memory. Programs are already stored in ROM, and may be run immediately. For example, your microwave oven lets you choose different settings and temperatures. You did not have to program these settings after you purchased your microwave oven, because they are permanently stored in ROM.

Your ST operating system is already in ROM. However, programming languages and applications must be loaded in from diskette.



You can use and read memory in GEM and TOS, but you can't change the memory, which has already been set up by developers and programmers.

In the preproduction development of computers, ROMs are not used immediately (the production costs would be far too high using incorrectly-programmed ROMs).

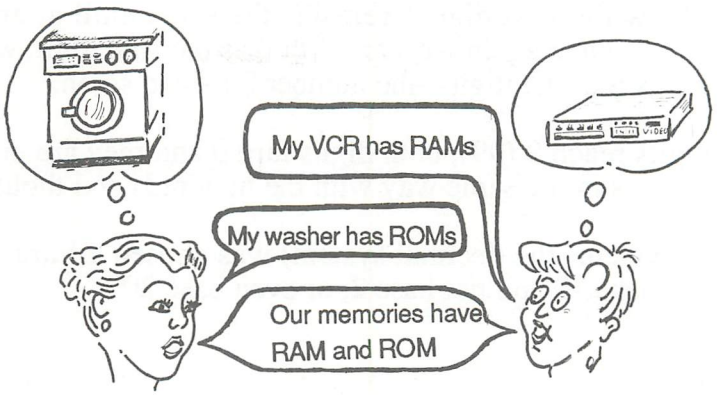
Therefore, programmers use the *EPROM* (Erasable Programmable ROM). EPROMs let the user determine the structure of the program code himself, i.e., "burn" an EPROM (install a program). If the results are unacceptable, he can erase the old information under ultraviolet light. With the proper equipment, you may "burn" EPROMs with the ST yourself. You could store a self–written program on the EPROM (for example, the telephone directory program in this book). You would never have to load this program from disk—the program on an EPROM would be up and running the moment you turned on the computer.

EPROMs are easily recognizable because they usually have small strips of paper on their top side. The size of the chip is determined by the number of

pins coming out of the chip. This strip of paper protects it from being erased or otherwise damaged. If this paper is removed and the chip is held under ultraviolet light, it will be erased, and then can be reprogrammed.

The *PROM* is similar to the EPROM except that a PROM cannot be erased. Once a PROM is burned, its contents cannot be changed. The only way to change programming is to replace the old PROM with a newly–burned PROM.

The most important chips in a computer are the *RAMs*. They can memorize all data entered, and can also immediately lose them due to the NEW command or power failure.



RAM stands for Random Access Memory, which means that the memory can be directly accessed. Of course, we also have free access to ROM, but we cannot change ROM for our personal use, as we can with RAM.

The RAM is our freely programmable memory. We can save our data or programs in RAM. But to keep a permanent copy of this data, we have to transfer the information from RAM to magnetic media, such as disks or hard disks.

5.1.3 It's as easy as 1, 10, 11

The Zuse computer worked through relays which read two conditions—"current on" or "current off"—and displayed the result by lighting incandescent bulbs. How can a computer work with these two conditions?

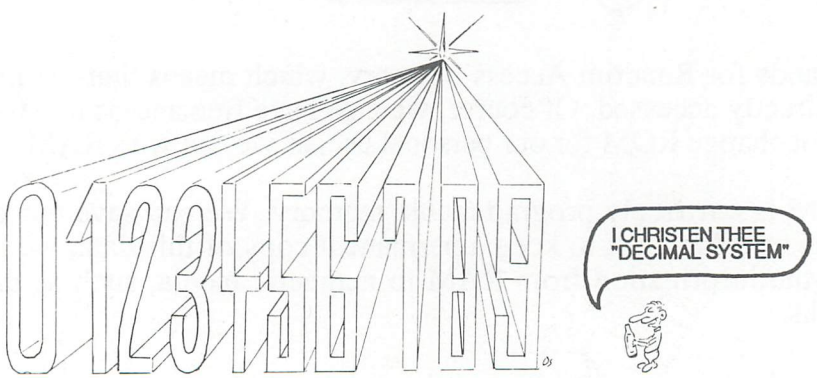
In everyday work we use a number system of ten different figures:

0 1 2 3 4 5 6 7 8 9

The lowest number is 0 to the left, and 9 at the right. Then what? You start again from the very beginning, but the 0 is now preceded by a 1—the number 10. Now the first digit 1 remains the same until after the second digit reaches a 9. Then again we start with 0 as the first digit, which is now preceded by 2 as the first digit—the number 20—and so on.

Once both digits reach 9 (99), both digits turn 0 and they are preceded by a 1(100). It happens in the same way with the hundreds and thousands.

This number system (the decimal system) was chosen arbitrarily. Why do we work from base 10 and not base 2, or even base 20?

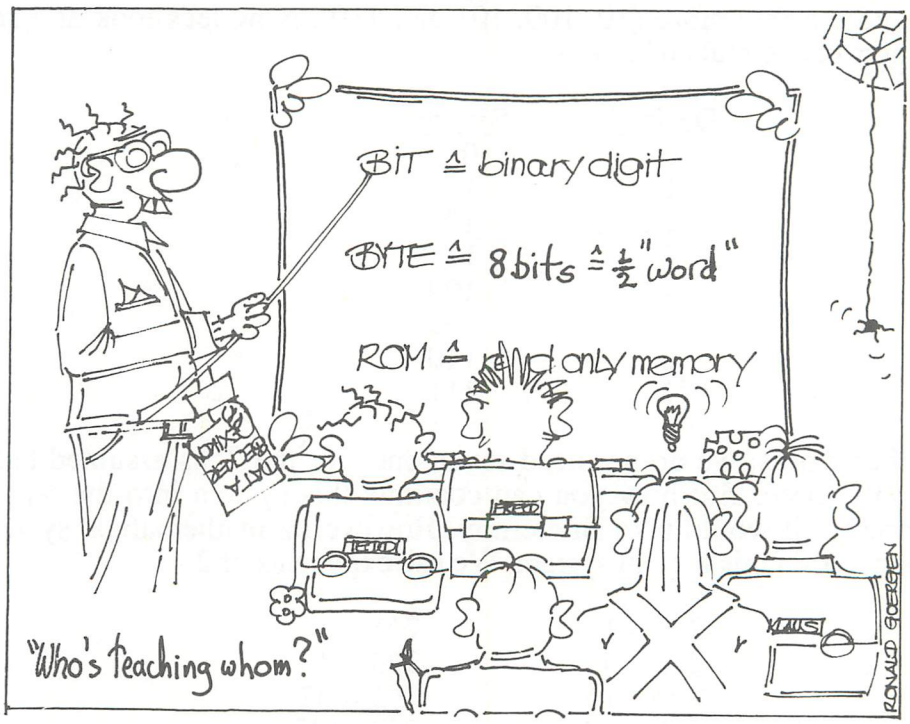


Once computers were developed, the next consideration was how it should learn a number system, especially the decimal system. You cannot teach the decimal system to the computer, as computer pioneers found out.

Switches were developed that could differentiate more than two conditions, depending on the intensity of the electricity. However, the computer didn't always recognize the different degrees of electrical current. The result was incorrect computations and incorrect results.

The final consensus was that the computer should only use on and off settings. Accurate system functions were almost guaranteed. A name for this off/on (0 and 1) mathematical system was chosen some time ago by mathematicians during the development of the decimal system. It is known as the *binary system.*

How is it possible to process larger numbers with only the digits 0 and 1? Well, you could use all ones (1). For example, for the number 10, ten ones, and three hundred ones for the number 300. This is possible, but we want to use 0's as well as 1's.



Let's look at the binary system the same way as with the decimal system. We have two numbers which can be connected to each other with increasing size—0 and 1. Unlike the decimal system, 1 is our largest number. Therefore, once we get past counting 0, 1, the next number will be 10. In the binary system, this figure is not read as "ten", but as "one–zero".

Next we raise the right digit by 1, and we get the binary number 11 ("one-one"). Again the number has reached its highest value before adding a left digit (similar to 99 decimal). The next number would be 100, then 101, 110 and 111.

The binary system may appear complicated at first, but if you proceed here in exactly the same way as you already do in decimal, you'll quickly catch on. If one digit in the decimal system surpasses its highest value (greater than 9), a 1 is simply added to its left.

To translate larger decimal numbers into binary numbers, we first assumed that we had to take the respective quantity of ones for each figure, i.e., ten ones for one–zero. But if we look at our sequence from 0 through 111, we'll find that in between we not only computed the conditions 1 and 11 but also quite a few more (10, 100, 101 and 110). Now let's look at a set of numbers in decimal and binary:

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

In other words, we do not need seven ones for the 7, as assumed before, but only three. But how you can continue this system into the tens and hundreds—it would take an eternity. However, a mathematical system is hidden within this logical system. We use exponents of 2:

2^0 = 1	2^1 = 2
2^2 = 4	2^3 = 8
2^4 = 16	2^5 = 32
2^6 = 64	2^7 =128

Now we fit each binary digit into the following table:

2^5	2^4	2^3	2^2	2^1	2^0
=32	=16	=8	=4	=2	=1
		1	1	0	0

= 8 + 4 + 0 + 0 = 12

Now add according to these powers of 2. The binary number 1100 results in 4 powers of 2, because only the third and fourth digit contain a 1; the added two exponents amount to 12. A further example:

Binary 111=decimal 2^2 + 2^1 + 2^0 = 4 + 2 + 1 = 7

Maybe you can also see how you can change 255 decimal to eight binary numbers, and 65535 decimal to 16 binary numbers.

It would be advantageous later on if you familiarized yourself with the binary system now—you'll run into this number system working with your ST and any other computer you might use. All computer logic is based on the binary number system.

In addition to the binary system and the decimal system, there is another number system in which eight different conditions exist: the *octal system*:

0 1 2 3 4 5 6 7 = the octal system

Yet another number system has sixteen conditions (0-9, then A-F): the *hexadecimal system:*

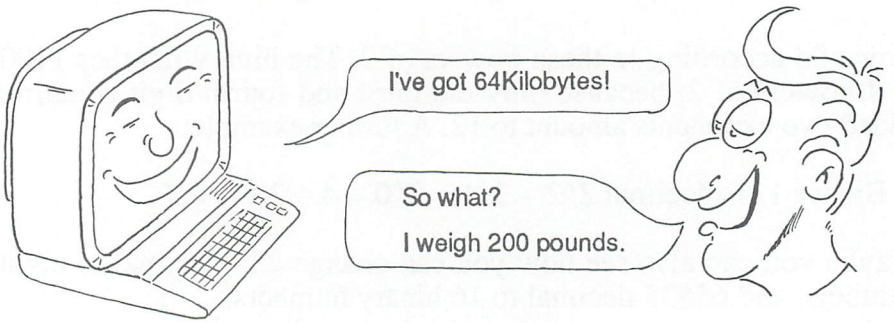
0 1 2 3 4 5 6 7 8 9 A B C D E F = hexadecimal system

The logic of these systems functions the same as with binary and decimal.

You'll find a set of BASIC programs and commands in the Appendix which will help you convert from one number system to another.

5.1.4 What is 512K, anyway?

As you know, RAM is the memory in your computer in which you store your programs, texts, etc. If you turn off your ST, or if the electric power is somehow interrupted, all information in RAM disappears. (Some computers have a battery backup power supply for RAM, so that if the current fails, the computer receives the electrical current for the RAMs from a battery).



The 520ST has 512 kilobytes (or 512K) of available memory for programming. The 1040 ST has twice that amount of memory. Other computers may have 3.5K, 64K, 128K, or 256K of available memory. What do these figures represent?

The smallest measurable memory unit is the *bit*, the abbreviation for binary digit. In the last section we learned that computers think in binary numbers, and that only two conditions are recognizable in computer logic: on and off. If a bit has a value of 1, that bit is on; a bit with a value of 0 is off.

The next memory unit is a *byte*, which is made of eight bits combined. A byte handles numbers from 0 (or 00000000 in binary—note the eight digits, representing the eight bits) to 255 (11111111 in binary). The standard unit for measuring memory capacity is the *kilobyte*, which consists of 1024 bytes. Broken down into its component numbers:

1 kilobyte = 1024 bytes (x 8 [bits per byte]) = 8192 bits

The 520ST, as mentioned earlier, has a capacity of 512K. If we do a little creative arithmetic, we can determine the number of bytes and bits that this memory occupies:

5.1.5 The function of the microprocessor

We have learned to interpret our memory size, and we know what RAM and ROM are, but that's not quite enough. Some kind of central "brain" must manage the computer's memory cells and the decimal/binary conversions.

This device is called a *microprocessor*. This is also called the *central processing unit*, or *CPU*. The CPU originally referred to the microprocessor only, but today the term CPU applies to RAM, ROM and microprocessor combined.

The microprocessor is a chip containing not only ROM (its own machine language instructions), but also RAM (a number of registers which process and coordinate all information in the computer). The microprocessor built into your ST is the Motorola 68000, a very fast, efficient and widely-used microprocessor.

You'll remember the last chapter, when we did calculations with zeros and ones, and when we mentioned writing 255 in binary with eight ones, and 65535 with sixteen ones. The figures 1 and 0 are called *bits*.

The 68000 is a 16/32-bit microprocessor, because it processes all at one time binary numbers up to (decimal) 65535. Many common home computers use 8-bit microprocessors like the Z80A found in the Commodore 128. The Z80A is not as efficient as the 68000, and considerably more difficult to program than the 68000. (The Z80 has 600 commands, while the 68000 has only 78 commands).

While the 68000 can process data in lengths of 16 bits (decimal 0 to 65535), its address bus has 24 lines (24 bits = decimal figures from 0 to 16777215). This makes the 512K of memory possible. If we were limited to 16 address lines, we could "only" access 65535 bytes.

512 [K] x 1024 [bytes per kilobyte] = 524,288 [bytes]
x 8 [bits per byte] = 4,194,304 bits

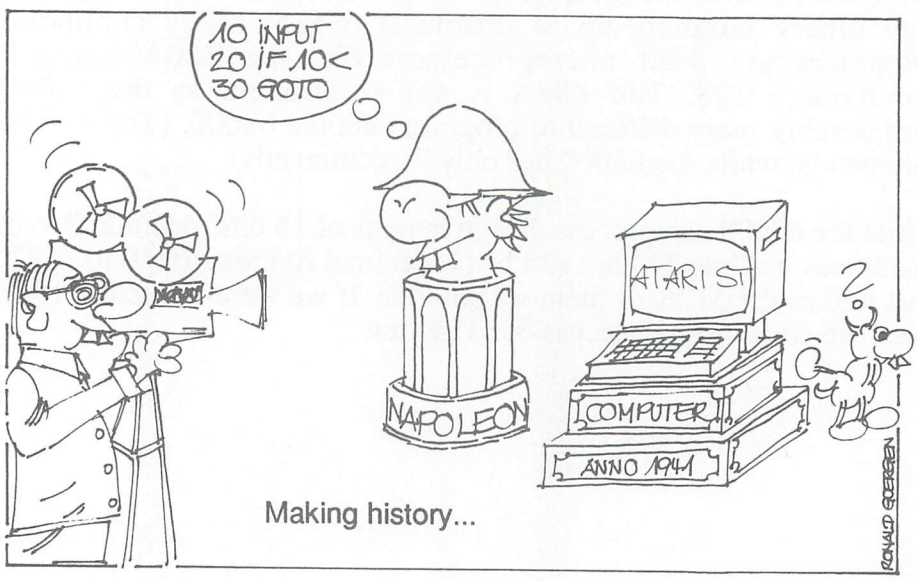
The 1040ST's memory capacity can be broken down in the same way:

1024 [K] x 1024 [bytes per kilobyte] = 1,048,576 [bytes]
x 8 [bits per byte] = 8,388,608 [bits]

Although 524,288 bytes and 1,048,576 bytes are more accurate figures, the computer world finds "512K" or "1024K" much easier to read, write and say, so kilobytes are the standard for memory measurement.

Remember, this capacity refers to the size of the RAM memory—the memory that we can actually use. A single character uses one byte of memory. This book is approximately 270,000 characters in length, so we would be able to comfortably fit the text of this book and a word processing program into the 520ST.

Let's take this a step further. There are approximately 2000 characters on a standard sheet of paper—this means that about 260 pages of text would completely fill a 520ST.



Two items of interest:

- The next generation of personal computers, such as the Macintosh II and Compaq 386, have 32–bit chips. A 32–bit microprocessor can simultaneously process numbers from 0 up to over 4 billion, and works far more efficiently than a 16–bit microprocessor like our 68000, which simultaneously addresses numbers from 0 to 65535.

- Another part of the microprocessor you should try to become familiar with is the ALU—the Arithmetic Logic Unit. The ALU is the actual computer within the microprocessor that performs the mathematical operations. We communicate with the microprocessor through the computer and its programming language. Its language is nothing like BASIC and LOGO, which consist of commands like `PRINT` for writing, `LIST` for listing a program, or `FORWARD` for moving the turtle.

  Since BASIC and LOGO are very simple for a user to learn, they are called *high-level programming languages*. If you want to program the microprocessor directly, you need much more knowledge, since a number of commands in the 68000's own language are required just to perform one BASIC or LOGO command. People program in the microprocessor's language—*machine language*—because this language executes hundreds of times faster than BASIC or LOGO.

5.2 Peripherals, options and possibilities for the ST

Now you know quite a lot about your ST's internals. You might have already thought about how to expand and improve your computer's performance.

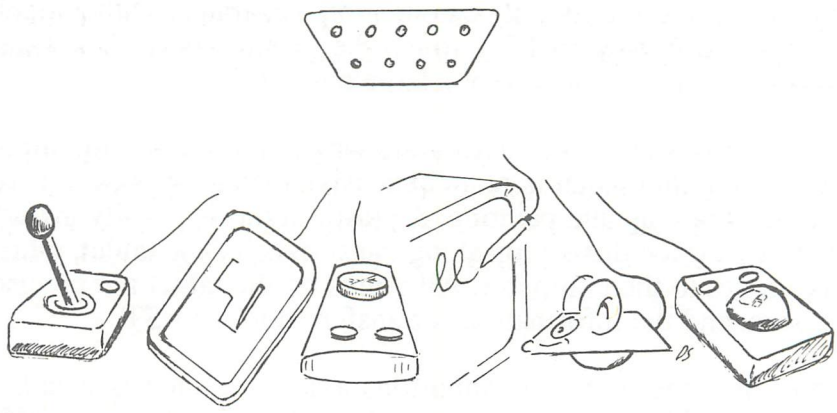
The ST has several ports that make it possible to interface with almost every piece of digital equipment on the market. There are also complete programs for the ST which you can use for word processing, data processing, games and more. Here are some ideas on what else you can do with your computer.

5.2.1 Ready for connection

As you already know, a computer consists of many components, including ROM, RAM and the microprocessor. ROM contains information that cannot be deleted, while RAM has memory available for programs. The microprocessor is the "brain" of the ST that is responsible for communication between RAM and ROM—and communication with the outside world.

Unless the ST can accept input from you through devices like the keyboard, and then output meaningful information to devices like the monitor screen, your computer is pretty worthless. The ST features several input and output connections known as *ports* that connect other components such as modems and printers to your ST. These components, all controlled by the ST's microprocessor, are generally known as *peripherals* . The following pages will give you some ideas for interfacing peripheral devices to the ST.

5.2.1.1 The joystick ports



The 520ST has two ports on its right-hand side into which you can interface two joysticks. The 1040ST's joystick ports are underneath the computer, as mentioned in Chapter 1. We can connect joysticks, paddles, lightpens, mice, trackballs and even a touch tablet to these ports.

Joystick

The joystick is probably the most easily recognized of all the devices listed above. Joystick movement is read from the primary directions (north, south, east and west), as well as directions in between (northeast, southeast, northwest and southwest). Put another way, joysticks are read in 45° increments, or a total of eight directions.

By moving the joystick in the desired direction, you move the object on the screen in the direction desired. You can use the joystick and fire buttons for an incredible array of video game software marketed for the Atari ST. Other interesting applications of joysticks include graphic software that lets you paint and draw a picture on the screen and then save it on diskette by pressing the fire button. We'll leave ideas for other joystick applications up to you.

Touch tablet

The previous example suggested that the joystick can be used to paint a picture on the screen. This can be done more accurately with a touch tablet, which you connect to your ST through the joystick ports. The tablet gives you a workspace about the size of a large postcard.

A touch tablet is a much more accurate way to record graphic input than a joystick. Although a tablet is more expensive than a joystick, it is designed expressly for drawing and painting. A plastic stylus is usually included with the tablet. You trace this stylus along the surface of the tablet, which sends each drawn dot to the computer. All points on the tablet are precisely read with sensors, and the information is transferred to your ST.

The tablet has one or two pushbuttons available. One is usually on the housing of the tablet itself, the other in the stylus. Tablets which offer menu selections, like the touch screens on some computer terminals, are expected to be introduced soon. You touch the stylus to the spot where the desired function is located (for example, save to diskette, or select a color).

Paddle

The paddle is a close relative of the joystick. The paddle consists of a potentiometer or "knob" which you turn to guide an object. It also connects to the joystick ports.

Here are two examples of how a program uses a paddle:

- The knob is used like a steering wheel to steer a car on the screen. Pressing the fire button on one side of the paddle speeds up the car; the other button slows down the car.

- A popular video game has a little man bouncing on a seesaw. If the little man hits the seesaw in the right spot, he is thrown high into the air, hitting a balloon in the sky with his head. The little man then falls back to Earth. You move the knob so that the little man hits the seesaw again instead of the ground.

Both games could be played with a joystick, but that's not as practical (or as fun) as using paddles for the same purposes.

Trackball

A trackball is a small box that contains a palm-sized, hard plastic ball. The ball protrudes through the top of the housing, and can easily be "rolled" in any direction. The trackball is read by the ST the same way as it reads a joystick (8 directions), but a trackball makes it much easier to position the cursor on the screen, for example.

Mouse

The mouse connected to the ST is basically a trackball turned upside down, contained in a small plastic housing about the size of a bar of soap. The mouse ball is moved by sliding the mouse housing on a hard surface.

The mouse, like the trackball, has at least one control button which calls functions like the joystick fire button. If the mouse is connected to joystick port 0, the two buttons on its upper side can be read for different functions.

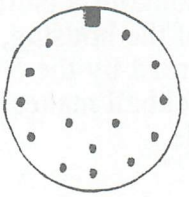
Lightpen

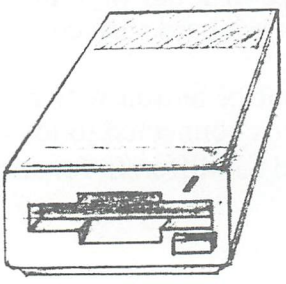
The lightpen is a graphics tool that plugs into the joystick port. But lightpens do not use an intermediate connection to the screen (like the touch tablet). You draw directly onto the ST monitor's screen. A photoelectric cell is located in the lightpen's tip. When the pen tip is held against the monitor screen, you can draw on the screen just as you would draw on paper with a regular pen.

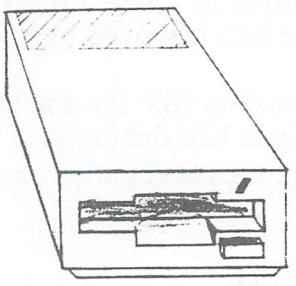
While the lightpen works very well on a computer monitor like the SM 124 or (color) SC 1224, the situation is very different when you're working on a regular television. The thick glass of a television screen refracts light, and the lightpen may not be as accurate as it could be. The results of lightpen input are also better with a color monitor than with monochrome.

There are many uses for the joystick ports, if you apply some thought to their use. But input devices cannot merely be plugged in—they require a *control program* of some sort. Games usually have built-in routines for joysticks or trackballs. Drawing programs accommodate lightpens and touch tablets. A mouse is normally used in operating systems and productivity programs (such as word processing). None of these devices will function without a control routine.

5.2.1.2 Disk drive connection



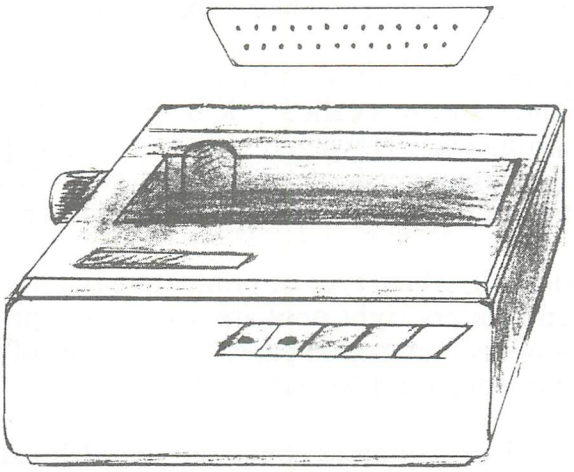




As you know, you connect a disk drive to your ST that uses 3 1/2" diskettes, and can then connect a second disk drive to the first drive (see Chapter 1 for complete connection instructions). The disk drives for the ST have storage capacities of 400K and 800K.

In addition, several third-party manufacturers make 5 1/2" IBM-compatible disk drives that can be connected to the ST. With one of these disk drives connected it is very easy to exchange data files (not programs) between the IBM PC and your ST.

5.2.1.3 Printer interface and operation



A Centronics parallel interface port is built into the Atari ST for printer interfacing and connection. Centronics is the name of the firm that developed this interface.

You may be wondering why the Centronics is a *parallel* port. Think back to the section on bits and bytes. You'll remember that one byte is made of up 8 bits. The ST always sends 8 bits at a time through the Centronics interface—that is, 8 electrical impulses to the printer, whether these impulses signal off or on. The process of sending 8 bits at a time is called *parallel transmission.*

We just read that all numbers in the decimal system can also exist in binary form. In much the same way, characters sent to the printer are represented by a code number, which is then transmitted through the parallel interface to the printer as a binary number.

Another method of data transmission is *serial transmission*, or sending data one bit at a time. This method transmits data much more slowly than parallel transmission. Serial transmission will be explained further in section 5.2.1.7, which deals with the RS-232 port.

Therefore, your Centronics parallel interface makes very fast data transfer between your ST and a printer possible. Most microcomputers have a Centronics interface, because it is the standard for parallel ports.

The Centronics printer interface allows the printer to understand the character set of the ST. The symbols appearing on the printer are represented in number codes. For example, the number 65 represents the letter A, the number 32 stands for a blank space, etc.

Several years ago an agreement was reached regarding a standard character set for printers. The result was called ASCII (American Standard Code for Information Interchange). However, this standard applies only to the characters with the number codes 0 to 127. The codes from 128 to 255 are left to the discretion of the different computer and printer manufacturers.

You may be wondering which type of printer to buy for your ST. The most widely purchased printers right now are *dot-matrix* printers. They print every letter, symbol and number on paper with a series of pins that strike an inked ribbon in the sequence of signals received.

Just like with the ST or any other computer, on/off binary encoding is the basis for the dot-matrix printer's operation. A dot-matrix printer does not feature crisp lettering like that produced on a typewriter, but usually has a printing speed of at least 50 characters per second. The reason they are so fast is that few moving parts are needed to drive the pins onto the inked ribbon and paper. A regular typewriter or even a *daisywheel* printer, which works under the same principle as a typewriter, cannot move at this speed, unless it is an extremely high-quality model.

In addition to the dot-matrix and daisywheel printers, other types of printers include ink-jet printers, thermal printers and laser printers.

In principle, an *ink-jet* printer works like a dot-matrix printer, except that it uses tiny nozzles to spray ink on paper as points. Unlike dot-matrix or daisywheel models, which tend to make a lot of noise when printing, ink-jet printers are very quiet. They are also expensive in comparison to dot-matrix and daisywheel machines. The main disadvantage of this printer (other than price) is that it produces poor carbon copies on triplicate computer paper—the ink droplets do not hit the paper hard enough to make an impression on the last carbon.

Thermal printers are even quieter than ink-jet printers. The print head generates heat in the shape of the letter to be printed. This heat is pressed against heat-sensitive paper, leaving a mark on the paper. Depending on the type of paper used, a thermal printer can print text in a number of different colors (usually blue, black or red).

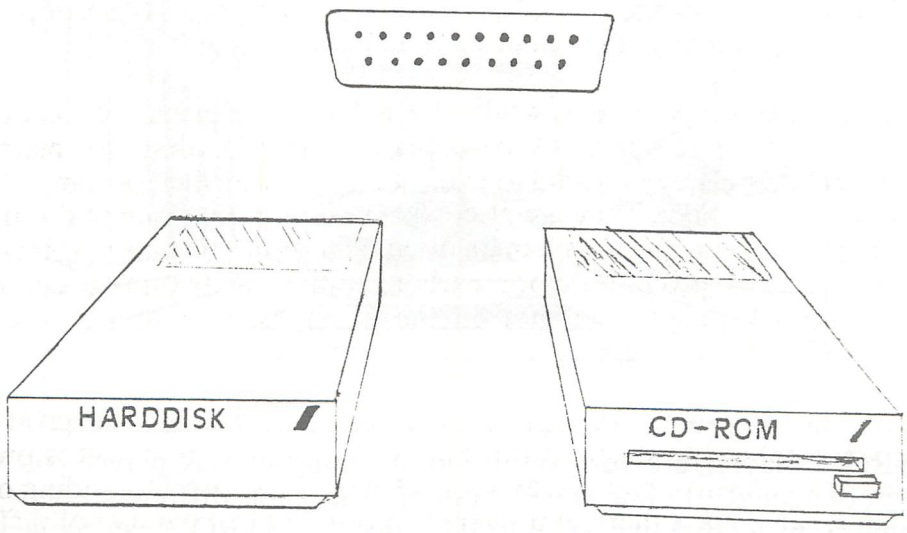
The disadvantage of the thermal printer is that the special paper is very expensive. If you print frequently, another type of printer would probably suit your needs better.

Finally, there is the *laser* printer. Laser printers produce an entire page at a time, usually within a few seconds. A laser printer uses a small semiconductor laser, lenses and mirrors to shoot pulses of light onto a rotating drum, which picks up toner in the spots where the light hits, and prints out the text. This book was printed on a laser printer. For you as a beginner, these printers are much too expensive, with starting prices of about $2000.

Almost any type or brand of printer can be connected to the ST because of its Centronics parallel port, which is an industry-standard interface. If you intend to purchase a printer, talk to your dealer or other ST users before you make a final decision.

5.2.1.4 The hard disk connection

In addition to the two disk drives you can hook up to your ST, you can also connect faster storage media like a hard disk drive and CD-ROM (Compact Disk-Read Only Memory).

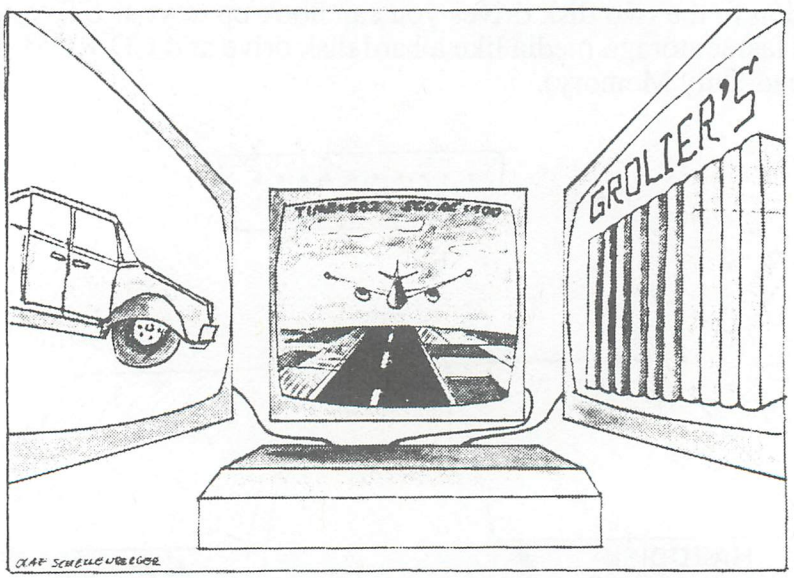


The *hard disk drive* works on the same principle as your floppy drive(s), except that a hard disk is mounted permanently inside the drive. Because the hard disk is airtight and permanently mounted, it can rotate at much higher speeds and work with much greater precision than a floppy disk drive. To give you an idea of the hard disk's speed: Floppy disks work with 250,000 bits per second. However, the Atari hard disk drive works with 10,000,000 bits per second (see *Baud* in Appendix C).

Connecting a hard disk

The hard disk drive is connected to your ST with a cable attached to the port next to the Centronics interface. The hard disk drive provides much more storage space than a floppy disk (10 megabytes and more). However, since the disk inside cannot be removed, you must delete data on the disk to make room for more when the disk has reached its capacity. Also, you cannot connect several hard disks in series as you could with floppy disk drives.

5.2.1.5 Connecting CD-ROM players

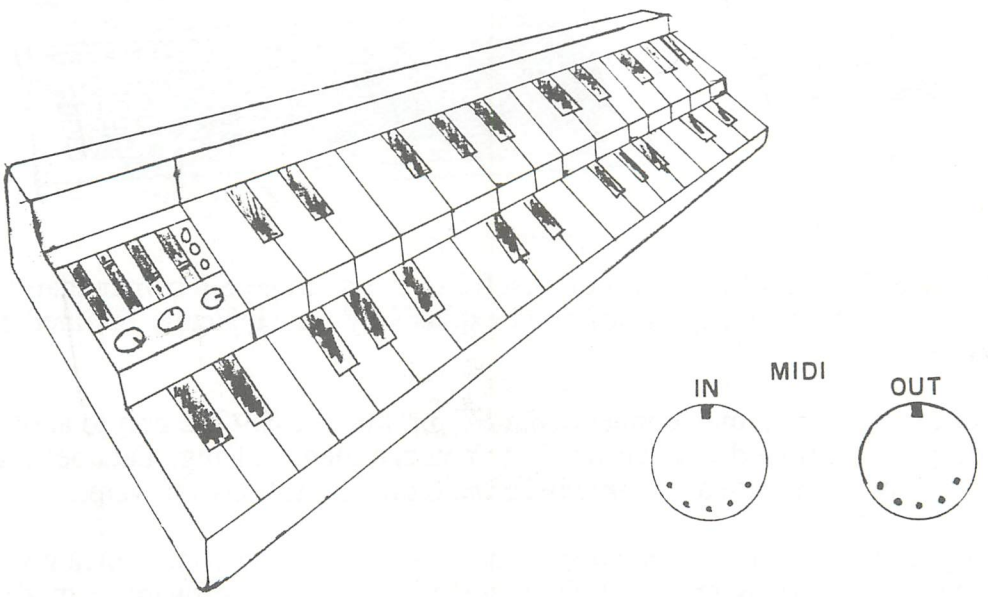


CD-ROM players are very similar to the compact disk player you might have connected to your stereo system. The main difference between a normal compact disk and a CD-ROM is that the CD-ROM can contain large amounts of text and graphic information, while your regular compact disk contains only musical information.

CD-ROMs are being used now in virtually every type of computer application. The one with which you may be most familiar is *Grolier's Encyclopedia.* The entire multi-volume set is stored on a single CD-ROM, complete with illustrations and a fast search capability. The Library of Congress has been cataloguing the nation's books and periodicals on CD-ROMs for years. Perhaps the CD-ROM application most valuable to the average person is the complete list of poisons and antidotes stored on CD-ROM in the National Poison Control Center. Any poison and its antidote can be called up in seconds, at a time when lost seconds can mean a lost life.

A CD-ROM player can be interfaced to your ST through the hard disk connection. However, at this writing CD-ROM players are priced much too expensively for the home computer user.

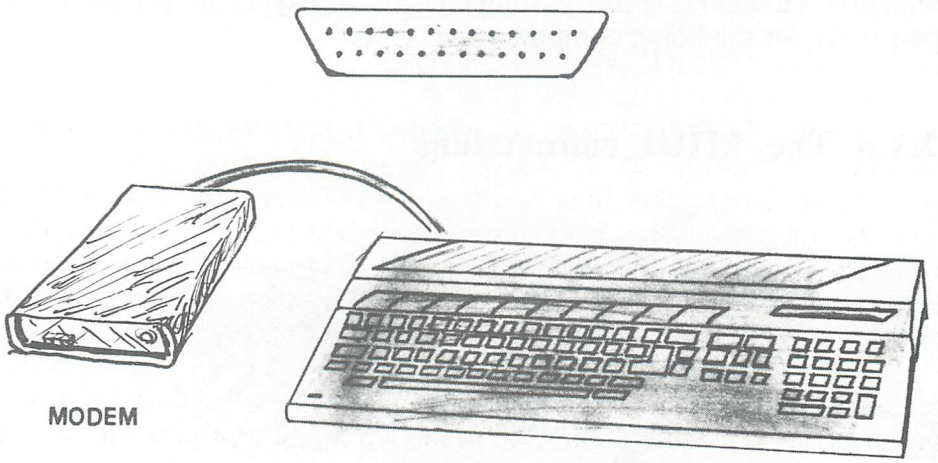
5.2.1.6 The MIDI connections



The MIDI (Musical Instrument Digital Interface) connections allow your ST to communicate with one or more MIDI-equipped digital music synthesizers. The MIDI IN and MIDI OUT connections allow the computer and instrument to "converse".

The ST can memorize different processes entered through the synthesizer, such as tone quality, musical passages or sequences of musical tones. These processes can be selected from the ST through a menu or the synthesizer, depending on the software used to control the instrument and computer. For complete background information, see Abacus Software's *ATARI ST Introduction to MIDI Programming* by Dorfman and Young.

5.2.1.7 The RS-232 interface



The RS-232 port is much more useful to the average ST owner than the MIDI interface, which is designed exclusively for musically inclined ST users.

For example, you may connect your ST through the RS-232 port to another computer equipped with an RS-232. You can then exchange data between both computers with a *terminal program* (communications software).

Or you might connect a *modem* to the RS-232 port and communicate with other computers by means of a telephone and terminal software. A modem (modulator/ demodulator) is a device that changes the electrical impulses in your computer to a form that can be transmitted over telephone lines, and convert computer signals received over the telephone into a form that can be understood by the terminal software. This method is used to communicate from terminal (computer) to terminal or terminal to bulletin board service

(BBS). A central computer known as a *mainframe* can even communicate with your ST and other computers all at once, from many miles away. You can also connect printers to your ST through the RS-232 port—but only printers that use a *serial* interface.

How is the RS-232 port different from the Centronics port? You'll remember that the Centronics interface is a *parallel* interface, sending data 1 byte (8 bits) at a time. The RS-232 is a *serial* port—that is, it transfers data one bit at a time in a long string, or series. Because of this, data transfer with the RS-232 port is considerably slower than with the Centronics interface.

5.2.1.8 The monitor connection

Your ST automatically recognizes whether it is connected to a color or a monochrome monitor. If it's a color monitor, you have a choice of two screen resolutions: medium resolution—640 by 200 *pixels* (picture elements, or individual points on the screen) and 4 colors; or low resolution—320 by 200 pixels and 16 colors. If you have a monochrome monitor connected, the only screen resolution available is high resolution—640 by 400 pixels, and no choice of colors (black or white).

The 520ST also can be connected to a standard television instead of a monitor through a cable connected to the RF modulator jack on the rear of the computer.

5.2.1.9 The power connection

Finally, there is a plug for the power supply. Be sure that the power cord fits snugly into this connection—you know the problems a sudden power loss can give you....

We hope that this section gave you some insight into what the ST's interfaces do, and that you're able to apply some of this knowledge later on.

5.2.2 Software

We'll now turn to a discussion of *software* (programs) that are presently available for the ST, or will be available in the near future.

5.2.2.1 Programming languages

We have some knowledge of BASIC and LOGO from Chapters 3 and 4. There are other, more efficient programming languages for the ST. One fine example is the language known as *C*. C is a difficult language to master, but it has many advantages over BASIC, not the least of which is speed.

If you have already mastered BASIC and now wish to learn C, there are a number of diskette-based versions of the language. They load into the ST the same way ST BASIC or ST LOGO loads. There are some exceptional books for learning the C language, such as *ATARI ST BASIC to C* , also available from Abacus.

There are many programming languages available to run on the Atari ST. Once you've mastered BASIC and LOGO, try writing programs in *machine language*. The ST's 68000 chip is easy to program, and is a fast and efficient microprocessor. If you are interested in learning machine language, there are a number of assemblers and books on the market (including the ***AssemPro*** assembler package and *ATARI ST Machine Language*, both from Abacus).

5.2.2.2 Application programs

Several commercial programs for the ST were on the store shelves even before the ST's introduction in 1985. Scores of commercial software packages are now available for word processing, data management, financial records, bookkeeping, drawing and painting, etc. You'll have to make the selection yourself, based on your own needs. It may be helpful to ask your Atari dealer for advice, or talk to other ST owners to assist you in your choices. Try to have the software demonstrated before you buy, so that you can get a clear idea of what you're getting for your money.

Some programs use the mouse and GEM, and some do not. For example, Digital Research, the developers of GEM, sell a word processor and a painting program (GEMwrite and GEMdraw, respectively) which take full advantage of GEM's capabilities. Other software publishers offer a complete line of quality applications, including Abacus' ***PaintPro***, ***TextPro***, ***DataRetrieve*** and ***PowerLedger***. Many games and educational programs have been written or converted for the ST computers, and more programs appear on the market every day.

It doesn't really matter whether we talk hardware or software. Owning an ST means that you own a computer for the future. We will close this chapter with a few words about the ST's past, as well as its future.

5.3 Epilogue: The significance of the Atari ST

At the January 1985 Consumer Electronics Show in Las Vegas, Atari president Jack Tramiel said, "We will not sell a home computer. We will not sell a business computer. But we WILL sell a personal computer."

What did he mean by that? There has never been a clear definition of the term *personal computer*. Is it a computer that does personal things, or is it personal in the sense that it is used exclusively at home? Or is it personal because it's small enough to carry around with you?

Although the term "personal computer" will be a subject for debate for a long time to come, the slogan Tramiel used to introduce the ST—"Power without the price"—is clear to everyone.

What is so new about the ST series of computers? First, you have at least 512K of memory. Add to that the Motorola MC68000 microprocessor, an extremely fast processor. Instead of handling only 100,000 mathematical operations per second, as with some "home" computers, the ST's 68000 microprocessor is capable of handling 1,000,000 calculations per second!

Since every computer solves problems logically with mathematics, you may correctly assume that the faster a computer "thinks", the better it is. Your ST can look up the "Computers" entry in your *Grolier's Encyclopedia* CD-ROM in a few seconds, where it might take you a few minutes to find the entry in the book version of *Grolier's*.

The ST system includes many devices at fairly low prices which you can connect to the ST, backing up the slogan "Power without the price."

Furthermore, your ST has several standard interfaces for connection to other computers, almost every kind of printer and even electronic musical instruments. Also, the ST's graphic capabilities are remarkable: 512 colors and a high-resolution screen of 640X400 pixels.

One other point in the ST's favor: Until now it was almost impossible for anyone to start computing from scratch without running into problems. One of the main reasons was that operating a computer required commands which seemed quite baffling to the average person ("Now, how do I start a program again: `RUN`, or `EXEC`, or `FORMAT A>`?").

Jack Tramiel had the right idea: he designed an icon-based operating system which would appeal to the new user. Anyone who can see can also understand pictures—this shows real consideration for the common man!

The concepts and technology behind the Tramiel Operating System and Graphics Environment Manager took years to develop. However, in 1984 Tramiel and Atari realized that it was time to offer these technological wonders at a price people could afford. Developmental work for the ST began in 1984. In January 1985, the 520ST computer was introduced to a startled public at the Consumer Electronics Show in Las Vegas, and it has been selling well ever since. More recently Atari has introduced one-, two- and four-megabyte models of the ST.

••• ••• ••• ••• •••

A serious disadvantage to writing computer books, and no one is more aware of it than authors and publishers, is the fact that by the time a book reaches the shelves of your bookstore or software store, chances are that changes have already been made to the computer or software. This means that some information in a new book may be outdated as soon as it's printed. For example, the early STs had their operating systems on diskette instead of in ROM: this meant that you had to load in TOS and GEM from diskette. As this book reached the editing stages, the ST was modified—suddenly everyone had TOS in ROM, and *The Atari ST for Beginners* was in need of a massive re-editing!

Computer companies are living, growing entities. They develop new ideas and concepts, improvements and enhancements, and add these to their machines. To keep track of the latest in computers, you should purchase a computer magazine regularly. Although there is still a small delay between writing and publication, magazines are the most up-to-date publications available.

The Appendices contain information any ST owner should know: Character set, number system conversions, etc. Appendix C contains a short glossary which defines the technical terms in this book. Finally, an index will help you find information and page references in this book.

We wish you all the best with your ST.

Appendices

Appendix A: The Atari ST character set

The ST has a comprehensive character set:

There are two ways to call the special characters (the ones not shown on the keyboard). You can output the characters with `PRINT CHR$()`, or you can access the characters directly from the keyboard:

CHR$	(1)	<Control><A>
CHR$	(2)	<Control><B>
CHR$	(4)	<Control><D>
CHR$	(5)	<Control><E>
CHR$	(6)	<Control><F>
CHR$	(9)	<Control><I> or <Tab>
CHR$	(11)	<Control><K>
CHR$	(12)	<Control><L> or <Control><Shift><+>
CHR$	(14)	<Control><N> or <Control><,>
CHR$	(15)	<Control><O>
CHR$	(16)	<Control><P> or <Control><0>
CHR$	(17)	<Control><Q> or <Control><1>
CHR$	(18)	<Control><R>
CHR$	(19)	<Control><S> or <Control><3>
CHR$	(20)	<Control><T> or <Control><4>
CHR$	(21)	<Control><U> or <Control><5>
CHR$	(22)	<Control><V>
CHR$	(23)	<Control><W> or <Control><7>
CHR$	(24)	<Control><X> or <Control><8>
CHR$	(25)	<Control><Y> or <Control><9>
CHR$	(27)	<Esc>
CHR$	(26)	<Control><Shift><less than (<)>
CHR$	(26)	<Control><Shift><0>
CHR$	(26)	<Control><6>
CHR$	(26)	<Control><->

If you want to display the Atari symbol on the screen, for example, you can do it from program mode in BASIC by using:

```
PRINT CHR$(14); CHR$(15)
```

Or you can call the symbol in direct mode by pressing the following keys:

Press <Control><N>, then press <Control><O>

Appendix B: Conversion programs

1. Converting decimal to binary

Problem: Convert 255 decimal to its binary equivalent

```
10 INPUT "DECIMAL NUMBER";A
20 FOR N=15 TO 0 STEP-1
30 IF A=INT(2^N) THEN A$=A$+"1":A=A-INT(2^N):Z=1
40 IF A>INT(2^N) THEN A$=A$+"1":A=A-INT(2^N):Z=1
50 IF Z=0 THEN A$=A$+"0"
60 Z=0:NEXT N
70 PRINT A$

RUN

DECIMAL NUMBER ? 255

RESULT:  0000000011111111
```

2. Converting binary to decimal

Problem: Convert 11111111 binary to its decimal equivalent

```
10 INPUT "BINARY NUMBER";A$
20 FOR N=1 TO LEN(A$)
30 IF MID$(A$,N,1)="1" THEN Z=1
40 IF Z=1 THEN Z=0:A=A+2^(LEN(A$)-N)
50 NEXT N
60 PRINT A

RUN

BINARY NUMBER? 11111111

RESULT: 255
```

3. Converting decimal to hexadecimal

Problem: Convert 255 decimal to its hexadecimal equivalent

```
PRINT HEX$(255)
```

```
RESULT: FF
```

4. Converting hexadecimal to decimal

Problem: Convert $FF hexadecimal to its decimal equivalent

```
PRINT &HFF
```

```
RESULT: 255
```

5. Converting decimal to octal

Problem: Convert 255 decimal to its octal equivalent

```
PRINT OCT$(255)
```

```
RESULT: 377
```

6. Converting octal to decimal

Problem: Convert 377 octal to its decimal equivalent

```
PRINT &O377
```

```
RESULT: 25
```

7. Converting binary to hexadecimal

Problem: Convert 11111111 binary to its hexadecimal equivalent

```
10 INPUT "BINARY NUMBER";A$
20 FOR N=1 TO LEN(A$)
30 IF MID$(A$,N,1)="1" THEN Z=1
```

```
40 IF Z=1 THEN Z=0:A=A+2^(LEN(A$)-N)
50 NEXT N
60 A%=A
70 PRINT HEX$(A%)

RUN

BINARY NUMBER? 11111111

RESULT: FF
```

8. Converting hexadecimal to binary

Problem: Convert $FF hexadecimal to its binary equivalent

```
10 INPUT "HEXADECIMAL NUMBER";A$
20 A$="&H"+A$
30 A=VAL(A$):A$=""
40 FOR N=15 TO 0 STEP-1
50 IF A=INT(2^N) THEN A$=A$+"1":A=A-INT(2^N):Z=1
60 IF A>INT(2^N) THEN A$=A$+"1":A=A-INT(2^N):Z=1
70 IF Z=0 THEN A$=A$+"0"
80 Z=0:NEXT N
90 PRINT A$

RUN

HEXADECIMAL NUMBER ? FF

RESULT: 0000000011111111
```

Appendix C: Mini glossary for computer users

Assembler:

Program used to access and program a computer in its native language (see **Machine language**)

BASIC:

High-level programming language that can be learned with relative ease, since commands are easily edited and errors can be found immediately. Considered to be one of the most popular programming languages ever written

Baud:

Speed of transfer from computer to peripherals, measured in bits per second; standard rates are 300, 1200, 4800 and 9600 baud

Binary system:

Number system using only 0's and 1's; computers use the binary system for all internal computations

Bit:

The smallest information unit in a computer (8 bits=1 byte=1 character)

Bug:

Program or hardware error; based on retired US Navy Admiral Grace Hopper's tale of an actual insect "crashing" a relay-driven computer in the 1940s

Byte:

Equal to 8 bits; 1 byte can be a letter or character. You gauge computer memory in 1 Kilobyte (1024 byte) increments

C:

High-speed programming language. Since error detection does not take place until compiling (conversion into machine language), the language is very difficult to learn (see also **BASIC**)

Centronics:

Company that developed the standard printer connection on your ST. This interface will connect directly to most printers (see **Compatibility**)

Chip:

One small electronic component containing a number of miniaturized components and circuits, created by a photographic process; also called integrated circuit

Compatibility:

Easy exchange of hardware and software between different types of computers

Control code:

Function performed by a control key, either in direct mode or in a program

CP/M:

Abbreviation for Control Program for Microprocessors; operating system used mainly for commercial software programs

CPU:

Central Processing Unit. The heart of the computer that controls all data flow; can be a single integrated circuit or a series of chips

Cursor:

Character on screen that shows where your text will appear

Database:

Software used for filing, mailing lists, recipes—anything that requires keeping track of files

Debugging:

Elimination of program errors (see **Bug**)

Decimal system:

Number system using the numerals 0 to 9

Diskette:

Mylar disk coated with a magnetic substance and enclosed in plastic case; used for data storage. Standard diskette sizes are 3", 3.5" and 5.25" (older diskettes came in 8" size)

Disk drive:

Device for loading and saving files on diskettes

Dot-matrix printer:

High-speed printer with a head consisting of a matrix of pins that print letters, characters and graphics on paper as a series of dots

Drop-down menu:

Menu that drops down when the mouse pointer is placed on a selection of the menu bar

Editing:

Correcting program errors, or typing in new program lines on the computer

EPROM:

Chip that retains data "burned" into it, but data can be erased by ultraviolet light for later re-burning

Error message:

Message displayed on the screen in response to any error; messages can be in text or numerical form

Fire button:

Button found on joysticks, paddles, etc.; used mostly in game control

Floppy diskette:

Flexible magnetic disk that stores data (see **Diskette**)

Function keys:

Ten keys arranged across the top of the ST keyboard as <F1>, <F2>, etc. Functions can be assigned by the user or by programs

GEM:

Graphics Environment Manager; graphic operating system of the ST which allows up to four windows onscreen at a time and sets up the mouse as an input device

Graphic:

Any diagram, drawing or artwork produced by a computer, or created by a human being using a computer as an artistic medium

Hard disk:

Device used for rapid storage and loading of programs. Unlike floppy disks, the hard disk is a self-contained, sealed unit that can store much more data and access data much faster than a floppy drive

Hardware:

All mechanical and electronic parts of the computer: disk drive, computer, etc. (see **Software**)

Hexadecimal system:

Number system consisting of 16 numerals/letters (0 to 9 and A to F); closely matches binary system in structure, and is used in machine language programming

Icon:

Small picture representing an object or action (e.g., ST program icons are drawn as blocks; disk drive icons look like file cabinets)

Input device:

Peripheral used to convey information from you to the computer; for example, keyboard, joysticks and mouse

Integrated circuit

(see **Chip**)

Interface:

Connection between computer and other devices (peripherals); the ST has a Centronics parallel interface for printer connection, and an RS-232 port for modem or serial printers

Interpreter:

"Go-between" for computer and programming languages; translates the programming language into machine language, which the computer understands. BASIC and LOGO are interpreted languages

Joystick:

Control device with movable stick and fire button, used mostly for games

Kilobit:

1024 bits=128 bytes; computer memory measurement

Kilobyte:

1024 bytes; computer memory measurement

Lightpen:

Pen-shaped device with a photocell at its tip; allows direct drawing on the monitor screen with the help of software designed for the purpose

LOGO:

Easy-to-learn interpreted programming language; uses graphically-oriented "turtle" and English-like commands

Machine language:

Programming language understood by the computer; commands in BASIC or LOGO are turned by the interpreter into machine language; machine language programs are much faster than LOGO or BASIC programs as there is no interpreter involved

MC68000:

Motorola's 16-bit microprocessor used as a CPU in the ST

Megabyte:

1024 kilobytes, or 1,048576 bytes; computer memory measurement unit

Memory:

Electronic "space" available in a computer for data storage; also, different types of memory storage: RAM, ROM, EPROM, PROM

Microprocessor

(see **CPU**)

MIDI:

Musical Instrument Digital Interface; connection in the ST that interfaces to one or more MIDI-equipped music synthesizers or samplers (see **Synthesizer** and **Sound**)

Modem:

Short for modulator/demodulator; allows transfer of data over telephone lines; can be a direct connection between computer and telephone jack or an acoustic modem, which sends and receives data acoustically over the telephone

Modulator:

A chip that creates a computer output so it can be viewed on a monitor or television

Monitor:

Special video screen for displaying data; offers higher resolution than a normal television set

Mouse:

Input device consisting of a small box with one or two buttons on top, and a small ball poking out the bottom; the onscreen cursor can be moved by placing the mouse on a table and moving it about

Octal system:

Number system made up of eight numerals (0 to 7); counted as 0 1 2 3 4 5 6 7 10 11 12 etc.; similar to binary and hexadecimal systems

Operating system:

Program or internal coding that regulates control between computer and peripherals; CP/M and GEM are operating systems

Output device:

Peripheral used for conveying information from the computer; for example, a monitor, television set or printer

Paddles:

Input device with potentiometer and fire button, used mostly for game control; unlike joysticks, paddles operate in only two directions

Parallel data transfer:

Binary data is transferred over a cable or interface eight bits at a time (see also **Serial data transfer**)

Peripheral:

All hardware connected to a computer, such as printers, disk drives, etc.

Printer:

Output device that prints computer data on paper (called a printout or hardcopy); various methods are used to print the data, e.g., dot-matrix, daisywheel, ink-jet, thermal and laser printing

Programming language:

Series of commands and instructions that allows the programmer to control the computer; examples include machine language, and higher-level languages such as BASIC, LOGO and C

PROM:

Programmable Read Only Memory; chip onto which data can be recorded, or "burned; unlike EPROMs, PROMs cannot be erased

RAM:

Random Access Memory; memory with directly alterable contents; contents remain in memory until electric power is turned off

Relay:

Electromechanical switch used to make early computer logic circuits; relays were later replaced by vacuum tubes, transistors and chips

RGB:

Abbreviation for Red-Green-Blue; refers to high-quality color monitors, or the interface to connect such a monitor to a computer

ROM:

Read Only Memory; ROMs do not lose their data in their memory, even when the computer power is turned off (see also **EPROM, PROM** and **RAM**)

RS-232 port:

Interface used with modems or printers; uses serial data transfer (see next entry)

Serial data transfer:

Data transfer occurring one bit at a time, i.e. in serial (see **Parallel data transfer**)

Software:

Anything used by the physical components of a computer to accomplish its processing (see **Peripherals, Hardware**); programming languages, operating systems and programs are all software

Sound:

In this context, waveforms created by a chip within an electronic device and converted to acoustical vibrations by a loudspeaker; the ST has a Yamaha YM-2149 sound chip built in. Also, the ST can control MIDI-equipped digital music synthesizers through its MIDI ports

Spreadsheet:

Program that displays a series of rows and columns and allows you to perform calculations, budgeting, "what if.." simulations

String:

Series of alphanumeric characters, placed in quotes (e.g., `"This is a string"`); a variable made of a string, designated by a dollar sign after the variable name (e.g., `A$="This is a string variable"`)

Synthesizer:

Electronic musical instrument capable of reproducing tones and sounds through a series of tone generators, filters and controls (see **MIDI**)

TOS:

Short for Tramiel Operating System, named after Atari's Jack Tramiel; TOS is your ST's operating system

Touch tablet:

Input device used for graphics; a plastic stylus or your finger draw on the surface of a touch-sensitive "pad"

Transistor:

Electronic component developed in the late 1940s; a descendant of relays and vacuum tubes, later replaced by chips

User-friendly:

Refers to computers or operating systems designed for ease of use by the new computer owner; TOS is a user-friendly operating system

Variable:

Data stored in a certain area of memory; a quantity which can contain a number of values

Window:

Adjustable subscreen on the ST, used in editing, reading disk directories, input, etc.; the ST can display up to four windows on the screen at a time

Word processor:

Software program that converts your ST into a writing tool; ease in editing, rewriting, text deletion and draft work make the word processor a great improvement over a typewriter

Appendix D: ST BASIC error messages

Here is a collection of the most important error messages encountered in ST BASIC, including explanations of each message:

1: *Undefined error.* The error is not understood by the ST. If the error doesn't match any other error known to the operating system, an error 1 will occur.

2: *Something is wrong.* This error message results mostly from a misspelled BASIC command, e.g., `GTO` instead of `GOTO`; correct the spelling to fix the error.

3: *`RETURN` statement needs matching `GOSUB`.* The `RETURN` must be preceded by a `GOSUB`. You must jump to a subroutine with `GOSUB` before going back to the main program with `RETURN`.

6: *Number too large.* The number input was less than 1E-19 (-10,000,000,000,000,000,000, or -10 quintillion) or more than 1E+18 (1,000,000,000,000,000,000, or 1 quintillion).

7: *Not enough memory.* There may not be enough memory available to run a program, even if you own a 1040ST. The reason for this error can be either too long a program, or too many variables in a program.

11: *You cannot divide by zero...* Just as in mathematics, the ST cannot do the impossible.

13: *Types of values do not match.* You cannot assign a string to a numerical variable. For example, A cannot equal "A". For this you must use a string variable, like A$.

15: *Strings cannot be more than 255 characters long.* If you wish to store a longer text in strings, you'll have to use multiple string variables.

17: *`CONT` works only in `BREAK` mode.* The `CONT`(inue) command works only if the program was stopped by a `BREAK` or by pressing <Control><G>, or if `END` appears in the program.

23: *Program line too long.* A single program line can have a maximum of 255 characters.

30: *Window number invalid.* You may only use numbers 0 to 3 with window commands (0=**EDIT**, 1=**LIST**, 2=**OUTPUT**, 3=**COMMAND**); any other values cause this error.

53: *File not found on disk drive specified.* The program name cannot be found on the disk. Look at the disk directory before loading a program with `Load "Name"`.

58: *File exists.* A file already stored on diskette has the same name as you had typed in under `Save`. You'll have to delete the old file from the diskette, use another diskette, or change the name of the file you wish to save.

61: *Disk full.* You'll have to either delete programs from this disk to free up some memory, or use another disk that has free space.

64: *Invalid file name.* A filename may consist of up to eight characters plus a three-character file extension.

99: *---BREAK---* This appears on the screen if you have stopped a running program with <Control><G>. You may continue the program run by typing in `CONT`.

103: *Invalid line number.* Line numbers may not be less than 0 or greater than 65529.

106: *Line number does not exist.* With `GOSUB` or `GOTO`, you may not jump to a line number that does not exist.

107: *Number too large for an integer.* Integer values can be no less than -32767 and no greater than +32767.

108: *Input data is not valid, restart input from first item.* If a numerical variable is expected in response to an `INPUT` command, you may not enter a string variable. You have to restart `INPUT` from the first item to be input.

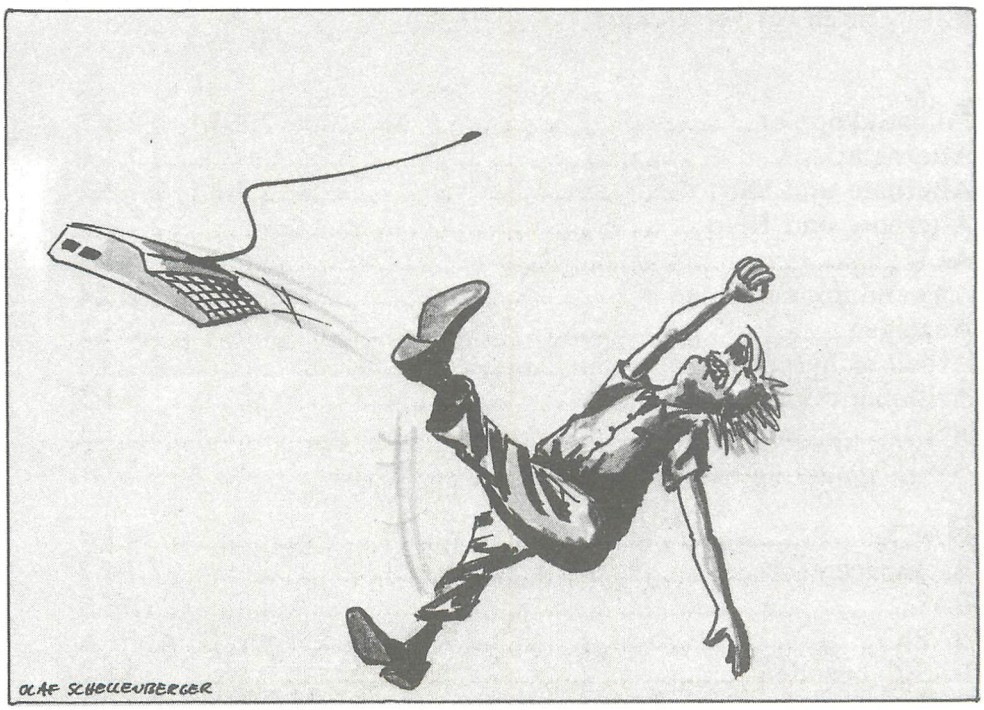
109: *STOP.* Program halt. The program may only be restarted with `RUN`.

204: *FOR statement needs a NEXT or WHILE needs a WEND.* A FOR...NEXT loop must consist of a FOR and a NEXT, or the program loop cannot be carried out successfully. A WHILE...WEND loop needs both a WHILE and a WEND for successful execution.

205: *NEXT statement needs a FOR or WEND needs a WHILE.* Program loops need both elements of each loop (see 205).

221: *System error...please restart.* There is only one thing you can do about this error—restart the system by pressing reset, or turn the computer off, then on again.

223: *Too many FOR loops.* You are not allowed to nest too many FOR...NEXT loops within one another.



Index

Catalog of Abacus Products for the ATARI® ST™

Abacus, P.O. Box 7219, Grand Rapids, MI 49510

Selected Books from our ATARI® ST™ Reference Library

GEM Programmer's Reference

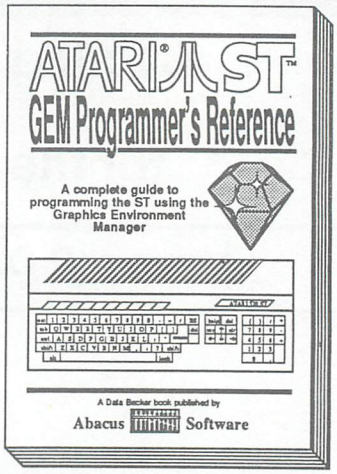
Atari ST GEM Programmer's Reference is an indispensable guide if you're a serious ST programmer needing detailed information on GEM. **Atari ST GEM Programmer's Reference** is written especially for the ST and has an easy-to-follow format. The GEM routines are explained with examples written in both C and 68000 assembly language.

Topics include:

- Overview of GEM: VDI, AES, GDOS, GIOS
- Intro to programming with GEM
- The ST Development System
- Using the Editor, C-compiler, Assembler and Linker
- Inside GEM: programming the Virtual Device Interface (VDI)
- Inside GEM: programming the Application Environment Services (AES)

GEM Programmers Reference is a complete programming handbook for all ST users. 412 pages. Optional diskette available.

GEM Programmer's Reference	Suggested Retail Price: **$19.95**
Optional Diskette	Suggested Retail Price: **$14.95**



"Anyone interested in learning how to manipulate the VDI or the AES will want to have this book at their fingertips..."

—Richard Kaller
ST Applications

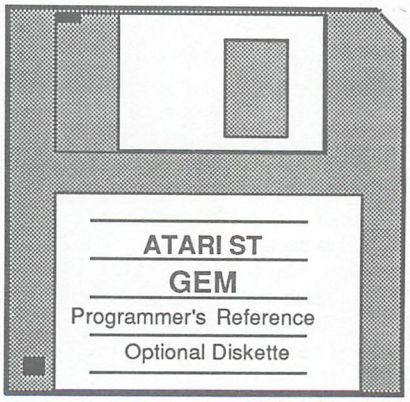
*"Despite its title, the Atari ST GEM **Programmer's Reference** is really a complete programming handbook for the ST."*

—Donald Evan Crabb
Byte

Optional Program Diskettes

Don't forget: optional program diskettes are available for all the books in the **Atari ST Reference Library** (except where noted). These optional diskettes contain all the program listings printed in the books, and will save you hours of tedious typing.

Each optional diskette: **$14.95**

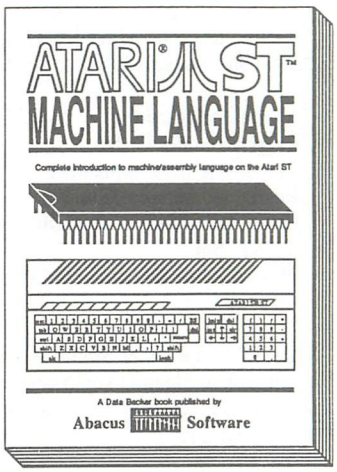


Selected Books from our ATARI® ST™ Reference Library

Machine Language

Atari ST Machine Language is the complete introduction to the high-speed world of 68000 machine language on the Atari ST. **Atari ST Machine Language** is required reading if you're interested in getting out the full potential built into the spectacular MC68000 microprocessor used in the Atari ST line of computers. Topics include:

- Logical operations and bit manipulations
- 68000 register structure and data organization
- Fundamentals of assembly language programming
- Operating system and programs
- Solutions to typical problems
- Program development
- Step by step programming
- Program and memory structure



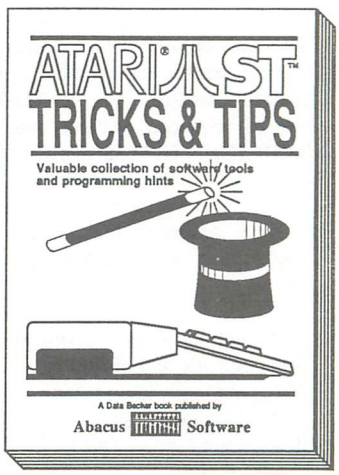
Atari ST Machine Language also contains many simple programs that progressively teach the fundamentals of programming in 68000 machine language. 280 pages. Optional diskette available.

Atari ST Machine Language Suggested Retail Price: **$19.95**
Optional Diskette Suggested Retail Price: **$14.95**

Tricks & Tips

Atari ST Tricks & Tips is a fantastic collection of ST programming tools and techniques that every ST user will find valuable. Teaches you how to define BASIC, assembler and C programs with the advanced programming techniques found exclusively in **Atari ST Tricks & Tips**. Topics include:

- Special ST BASIC commands
- "Safe" locations for M/L programs
- Using the VDISYS commands
- Mastering powerful GEM applications
- Producing fantastic graphics
- Building a RSC file



Program listings included in **Atari ST Tricks & Tips**:

- Super-fast RAM disk
- Time-saving printspooler
- Color print hardcopy
- Plotter output hardcopy
- Auto-starting TOS application
- Creating accessories

Plus much more—and all programs are included in the price of the book! Full four-color plates in appendix show you the ST's graphic capabilities. Fully indexed. 260 pages.

Atari ST Tricks & Tips Suggested Retail Price: **$19.95**
Optional Diskette Suggested Retail Price: **$14.95**

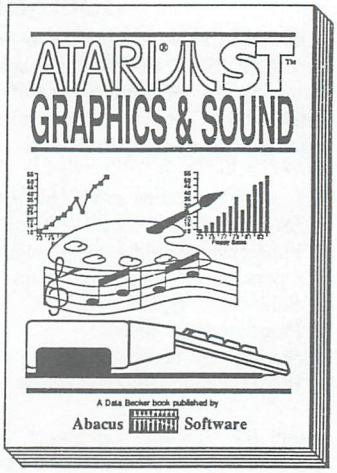
"...this book a best-seller and I can understand why."
—Pamela Rice Frank
Current Notes

Selected Books from our ATARI® ST™ Reference Library

Graphics & Sound

Atari ST Graphics & Sound teaches the ST user how to create graphics and make full use of the built-in sound capabilities of the Atari ST. Example programs listed in **Atari ST Graphics & Sound** are written in BASIC, C, LOGO and Modula-2. Topics include:

- Mirror and rotation
- Graphics under GEM
- Coordinate transformations
- Raster and vector graphics
- Principles of music synthesis
- Sound chip
- Plotting math functions in 2D & 3D
- Moire patterns
- Bar and pie charts
- Fractals
- Waveform generation
- The ST as a synthesizer
- MIDI control of musical devices



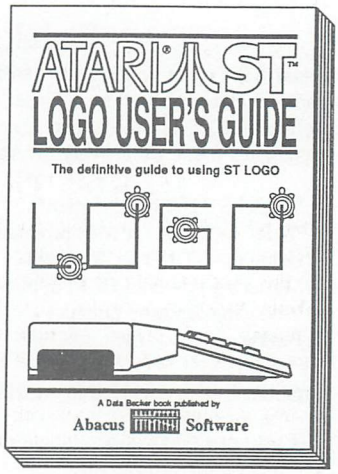
Atari ST Graphics & Sound is a must for the ST owner who wants an in-depth look at creating sophisicated graphics and surprising music and sound with the ST. 255 pages. Optional diskette contains dozens of graphics and sound programs.

Atari ST Graphics & Sound Suggested Retail Price: **$19.95**
Optional Diskette Suggested Retail Price: **$14.95**

LOGO User's Guide

ST LOGO was designed specifically to take full advantage of the Atari ST's fantastic graphic capabilities. LOGO's English-like words may be extra-ordinarily easy to learn, yet LOGO programs are actually built along the lines of advanced artificial intelligence languages like LISP. **Atari ST LOGO User's Guide** gently introduces the reader to the fundamentals of ST LOGO with numerous examples, dozens of actual screen illustrations and exercises that optimize the ST's features. Then it moves on to work with the more advanced features LOGO has to offer—readers will soon be programming highly complex tasks on their STs under LOGO. Topics covered:

- Thorough introduction to GEM, windows, and the mouse
- Randomizing and repetition
- Programming with recursion
- LOGO words & lists
- Data structures in LOGO
- Error output
- Computing with LOGO
- ST LOGO system, input and output commands
- Programs as lists



" A worthwhile addition to your reference library... contains many examples & demonstration programs in the LOGO language."
—Bruce Laubenheimer
Computer Shopper

Atari ST LOGO User's Guide covers everything your customers need to know about this acclaimed teaching and graphics language. 370 pages.

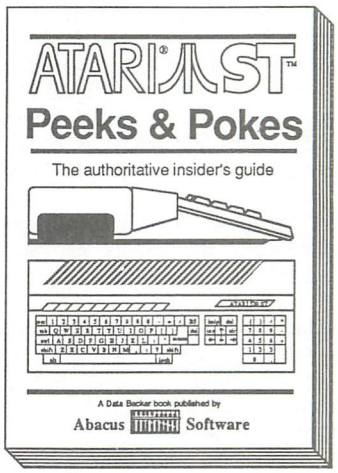
"For those folks who have been waiting for an excuse to start playing with LOGO, this may be it!"
—Steve Tearle
Atari Journal

Atari ST LOGO User's Guide Suggested Retail Price: **$19.95**
Optional Diskette Suggested Retail Price: **$14.95**

Selected Books from our ATARI® ST™ Reference Library

Peeks & Pokes

PEEK and POKE commands act as bridges between the user and the Atari ST's operating system through ST BASIC. **Atari ST Peeks & Pokes** enhances the user's knowledge of the ST and programs with numerous PEEK and POKE examples.



Atari ST Peeks & Pokes clearly explains a number of the most important PEEKs and POKEs and their application to common programming problems. At the same time, this book gives you an excellent look at the architecture and operation of the exciting Atari ST. Topics include:

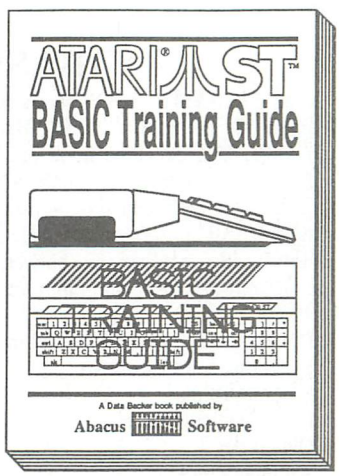
- The ST's configuration and interfaces
- The "intelligent" keyboard
- The mouse as a paintbrush
- Pointer and stack
- Customizing the desktop
- Important PEEKs & POKEs
- Making your own fill patterns
- ST communications
- Direct disk access
- Internal memory configuration

Atari ST Peeks & Pokes unlocks the secrets hidden within the ST with an excellent collection of "quick hitters" and information. 200 pages.

Atari ST Peeks & Pokes Suggested Retail Price: **$16.95**
Optional Diskette Suggested Retail Price: **$14.95**

BASIC Training Guide

Atari ST BASIC Training Guide for the Atari ST is a functional, educational and well-written introduction to ST BASIC. Quickly teaches you the fundamentals of programming with an introduction to program analysis, problem analysis, algorithms, and BASIC commands. This systematic book makes learning programming in the popular BASIC language quicker and easier than ever before.



Quizzes throughout the book help you learn to "think in BASIC" while you'r getting a practical grounding in the language. Topics include:

- Data flow and program flowcharts
- Advanced programming techniques
- Menus
- Multi-dimensional arrays
- Sort routines
- File management
- BASIC under GEM

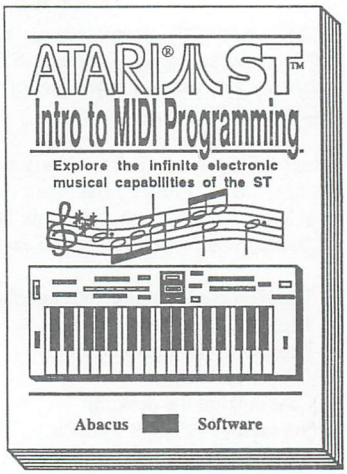
In addition, **Atari ST BASIC Training Guide** also contains advanced programming techniques if you already know ST BASIC. 312 pages.

BASIC Training Guide Suggested Retail Price: **$16.95**
Optional Diskette Suggested Retail Price: **$14.95**

*"The **Atari ST BASIC Training Guide** is a first-class text for ST BASIC users. It is clear, thorough, well-written and remarkably free of errors and typos... does a good jog of introducing the user to ST BASIC programming fundamentals. It also provides a valuable reference section for the more advanced user."*
—David Plotkin
Antic

Selected Books from our ATARI® ST™ Reference Library

Introduction to MIDI Programming



The digital music synthesizer is the musical instrument of the 80s. You can now buy synthesizers for under $1000 (as low as $250), play at least four voices at a time, and they can be connected to home computers through the Musical Instrument Digital Interface (MIDI) for computer control. The Atari ST is ideal for MIDI interfacing, since it has a built-in MIDI port. This means it's ready to hook up to any digital electronic musical instrument equipped with MIDI ports.

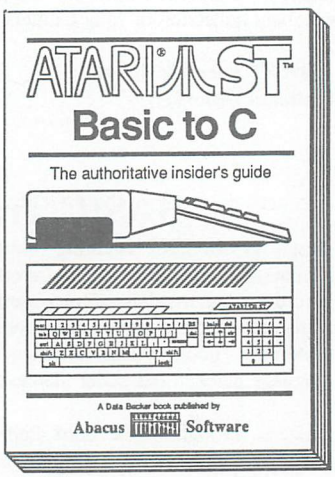
ST Introduction to MIDI Programming gives you the groundwork for discovering the infinite musical possibilities of the Atari ST's MIDI interface and your synthesizer. Topics include:

- Introduction to MIDI programming
- MIDI STANDARD and MIDI LANGUAGE
- Programming your synthesizer
- How to buy MIDI software
- Using the extended BIOS
- Source code from Xlent Software's ST MUSIC BOX© AUTO-PLAYER program
- C source codes for many programs and functions

Essential reading for anyone who uses the ST's MIDI port. 256 pages.

Introduction to MIDI programming Suggested Retail Price: **$19.95**
Optional Diskette Suggested Retail Price: **$14.95**

BASIC to C



Atari ST BASIC to C was written expressly for those of you who've learned the essentials of ST BASIC, but are hesitant to try another language. This excellent book quickly takes you beyond the BASICs and teaches how to program in the C language—the language of choice for thousands of advanced program developers. **Atari ST BASIC to C** places simple BASIC programs and their equivalents in C code side-by-side, with clearly-written comparisons between the two languages. Now you can learn the groundwork for C programming in only one day! Topics covered:

- Development, applications and the benefits of C
- Functions and text output
- Program format
- Loops and comments
- Data input
- Arithmetic in C
- Control structures
- Data types in C
- C pointers and arrays
- Common errors made by BASIC programmers

Atari ST BASIC to C skillfully guides the BASIC programmer through the necessary steps for programming in the C language. An essential addition to the libraries of all ST users.

Atari ST BASIC to C Suggested Retail Price: **$19.95**
Optional Diskette Suggested Retail Price: **$14.95**

"Imagine—if someone took all of the BASIC commands that you knew and loved so well, and showed how those commands would look and work in C...in a step-by-step, logical sequence with lots of examples—wouldn't that be nice? Well that's exactly what Abacus had Mr. Hartwig do, and it's very effective. This book creates an effective bridge between ST BASIC and the C programming language."

—David M. Pochron
The Atari Journal

Selected Books from our ATARI® ST™ Reference Library

3D Graphics

Teaches how to create impressive, lightning-fast three-dimensional graphics on the Atari ST in 68000 machine language. **Atari ST 3D Graphics** covers introductory concepts and background materials, graphic animation, using the assembler and much more.

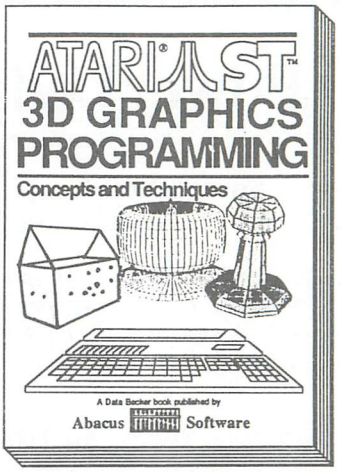
Learn real-time animation with dozens of graphic routines. 3D Graphics is an amazing book for all programmers interested in advanced level graphics.

Some of the topics covered include:

- Mathematical basis for 3D graphics
- Coordinate systems
- Scaling the axis
- Two- and three-dimensional transformations
- Hidden lines & surfaces
- Data structure for 3D objects
- Object animation
- Spatial projection
- Rotation of objects
- Light and shadows
- Introduction to 3D computer-aided design (CAD)

A must for all serious ST programmers. **Atari ST 3D Graphics** includes complete listings for a fascinating 3D pattern-maker and animator. 351 pages.

Atari ST 3D Graphics Suggested Retail Price: **$24.95**
Optional Diskette Suggested Retail Price: **$14.95**



The programs are clearly printed, well commented, planned in a sensible modular fashion, and contain many invaluable assembly-language 'tips and tricks.' And they work. ST programmers are fortunate to have this book."

—Douglas Weir
ST-Log

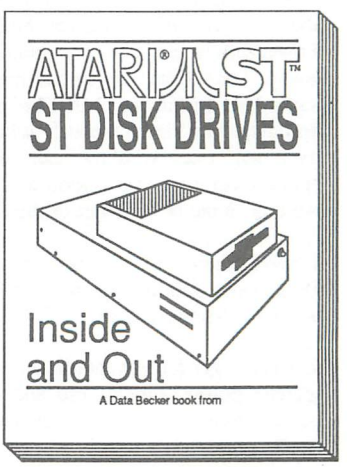
ST Disk Drives: Inside and Out

The latest title in the widely-acclaimed ***Abacus Atari ST Reference Library*** is the exclusive **Atari ST Disk Drives: Inside and Out.** This outstanding technical reference is the definitive source of information for the ST disk drives—it thoroughly discusses the floppy disk, the hard disk and RAM disk from both a programming and a technical perspective. In addition, the reader will find several full-length utilities and programming tools that enables him to further explore the ST disk drives' operations and capabilities. Topics include:

- Information of sequential and random access file structures
- Access to data files from BASIC, Pascal, C, and FORTRAN
- Data structures and management
- The boot sector and BIOS parameter bloc (BPB)
- The directory and File Allocation Table (FAT)
- Relocation table
- Hard disk format
- Details of drive construction: (DMA chip, disk controller, connector layout, and organization, etc.
- Command description, status interpretation, floppy interface, hard disk partition analyzer

Atari ST Disk Drives: Inside and Out is literally packed with utility programs. The book includes a complete listing for an easy-to-use RAM disk, BASIC/TOS interface, BASIC/FDC interface, BASIC loaders, Floppy-to-RAM disk copy, creating standard and foreign formats, and many more timesaving programs. **Available April '87.**

ST Disk Drives: Inside and Out Suggested Retail Price: **$24.95**
Optional Diskette Suggested Retail Price: **$14.95**



Selected Abacus Products for the ATARI® ST™

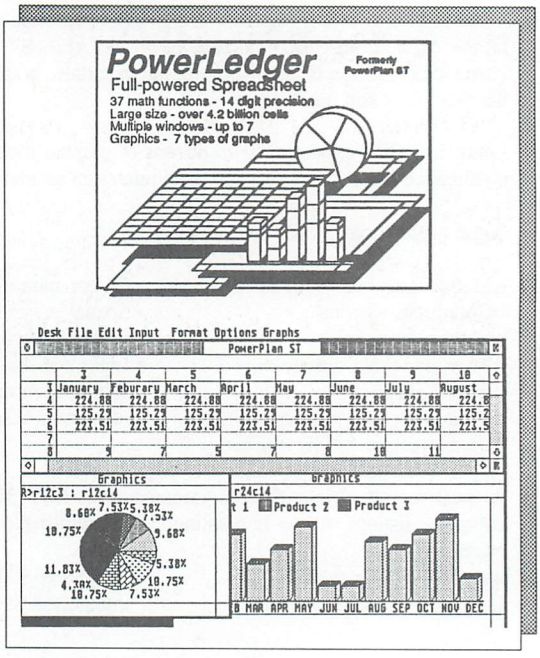
PowerLedger ST

(formerly PowerPlan ST)

Spreadsheet/Graphics package for the Atari ST

"A superior spreadsheet program for weekend bookeeping to the heavyweight job costing applications, (Powerledger ST) is a definite winner."

—Judi Lambert
ST World



Ever since VisiCalc and Lotus 1-2-3 stormed the personal computer market, the computer has become an important planning tool. **PowerLedger ST** brings the power of electronic spreadsheets to the Atari ST line of computers—it lets the user quickly perform hundreds of calculations and "what-if" analyses for business applications, and crunch raw data into meaningful, comprehensible information, to keep track of budgets, expenses and statistics.

PowerLedger ST is a powerful analysis package that features a large spreadsheet (65,536 X 65,536 cells—over 4 billion data items). It also contains a built-in calculator, online notepad, and integrated graphics.

PowerLedger ST is also very easy to learn, since it uses the familiar GEM features built into the ST. And PowerLedger ST can use multiple windows—up to seven. Data from the spreadsheet can be graphically summarized in in pie charts, bar graphs and line charts, and displayed simultaneously with the spreadsheet. For example, one window can display part of the spreadsheet; a second window a different part; and a third window, a pie or bar chart of the data.

PowerLedger ST works hand-in-hand with our **DataTrieve** data management package and our **TextPro** wordprocessing package.

PowerLedger ST's extraordinary combination of data and graphic power, ease of use and low price makes it a perfect tool for every ST owner's financial planning needs.

PowerLedger ST works with Atari ST systems with one or more single- or double-sided disk drives. Works with either monochrome or color ST monitors. Works with most popular dot-matrix printers (optional).

PowerLedger ST Features:

- Familiar drop-down menus make PowerPlan easy to learn and use
- Large capacity spreadsheet serves all the user's analysis needs
- Convenient built-in notepad documents your important memos
- Flexible online calculator gives you access to quick computations
- Powerful options such as cut, copy and paste operations speeds the user'swork
- Integrated graphics summarize hundreds of data items
- Draws pie, bar, 3D bar, line and area charts automatically (7 chart types)
- Multiple windows emphasize the user's analyses
- Accepts information from DataTrieve, our database management software
- Passes data to **TextPro** wordprocessing package
- Capacities: maximum of 65,535 rows
  maximum of 65,535 columns
  variable column width
  numeric precision of 14 digits
  maximum value 1.797693×10^{308}
  minimum value 2.2×10^{-308}
  37 built-in functions

PowerLedger ST Suggested Retail Price: **$79.95**

Chartpak ST

Professional-quality charts and graphs on the Atari ST

In the past few years, Roy Wainwright has earned a deserved reputation as a topnotch software author. **Chartpak ST** may well be his best work yet. **Chartpak ST** combines the features of his **Chartpak** programs for Commodore computers with the efficiency and power of GEM on the Atari ST.

Chartpak ST is a versatile package for the ST that lets the user make professional quality charts and graphs fast. Since it takes advantage of the ST's GEM functions, **Chartpak ST** combines speed and ease of use that was unimaginable til now.

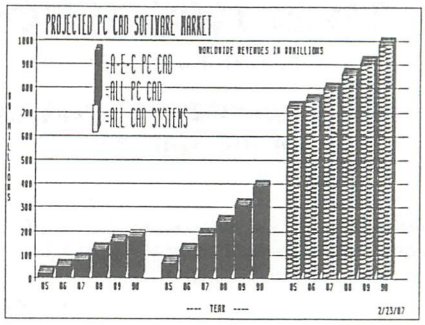
The user first inputs, saves and recalls his data using **Chartpak ST**'s menus, then defines the data positioning, scaling and labels. **Chartpak ST** also has routines for standard deviation, least squares and averaging if they are needed. Then, with a single command, your chart is drawn instantly in any of 8 different formats—and the user can change the format or resize it immediately to draw a different type of chart.

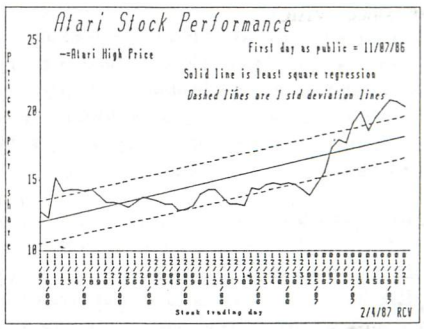
In addition to direct data input, **Chartpak ST** interfaces with ST spreadsheet programs spreadsheet programs (such as **PowerLedger ST**). Artwork can be imported from **PaintPro ST** or DEGAS. Hardcopy of the finshed graphic can be sent most dot-matrix printers. The results on both screen and paper are documents of truly professional quality.

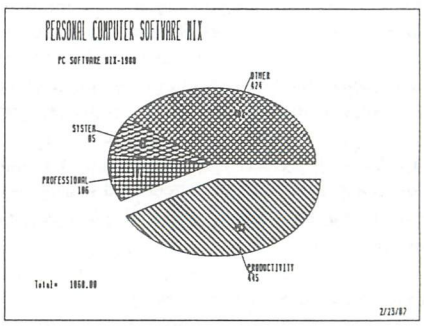
Your customers will be amazed by the versatile, powerful graphing and charting capabilities of **Chartpak ST** .

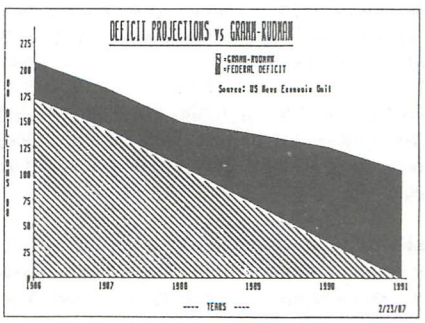
Chartpak ST works with Atari ST systems with one or more single- or double-sided disk drives. Works with either monochrome or color ST monitors. PWorks with most popular dot-matrix printers (optional).

Chartpak ST Suggested Retail Price: **$49.95**









Selected Abacus Products for the ATARI® ST™

DataRetrieve

(formerly FilePro ST)

Database management package for the Atari ST

*"**DataRetrieve** is the most versatile, and yet simple, data base manager available for the Atari 520ST/1040ST on the market to date."*

—Bruce Mittleman
Atari Journal

DataRetrieve is one of Abacus' best-selling software packages for the Atari ST computers—it's received highest ratings from many leading computer magazines. **DataRetrieve** is perfect for your customers who need a powerful, yet easy to use database system at a moderate price of $49.95.

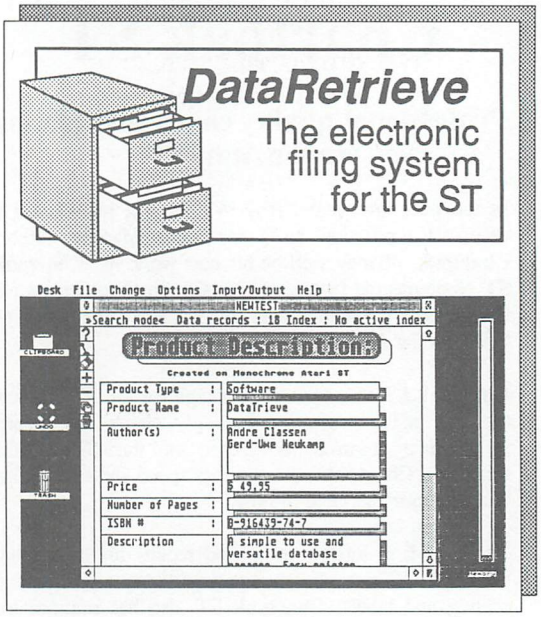
DataRetrieve's drop-down menus let the user quickly and easily define a file and enter information through screen templates. But even though it's easy to use, **DataRetrieve** is also powerful. **DataRetrieve** has fast search and sorting capabilities, a capacity of up to 64,000 records, and allows numeric values with up to 15 significant digits. **DataRetrieve** lets the user access data from up to four files simultaneously, indexes up to 20 different fields per file, supports multiple files, and has an integral editor for complete reporting capabilities.

DataRetrieve's screen templates are paintable for enhanced appearance on the screen and when printed, and data items may be displayed in multiple type styles and font sizes.

The package includes six predefined databases for mailing list, record/video albums, stamp and coin collection, recipes, home inventory and auto maintenance that users can customize to their own requirements. The templates may be printed on Rolodex cards, as well as 3 x 5 and 4 x 5 index cards. **DataRetrieve**'s built-in RAM disks support lightning-fast operation on the 1040ST. **DataRetrieve** interfaces to **TextPro** files, features easy printer control, many help screens, and a complete manual.

DataRetrieve works with Atari ST systems with one or more single- or double-sided disk drives. Works with either monochrome or color monitors. Printer optional.

DataRetrieve Suggested Retail Price: **$49.95**



DataRetrieve Features:

- Easily define your files using drop-down menus
- Design screen mask size to 5000 by 5000 pixels
- Choose from six font sizes and six text styles
- Add circles, boxes and lines to screen masks
- Fast search and sort capabilities
- Handles records up to 64,000 characters in length
- Organize files with up to 20 indexes
- Access up to four files simultaneously
- Cut, past and copy data to other files
- Change file definitions and format
- Create subsets of files
- Interfaces with **TextPro** files
- Complete built-in reporting capabilities
- Change setup to support virtually any printer
- Add header, footer and page number to reports
- Define printer masks for all reporting needs
- Send output to screen, printer, disk or modem
- Includes and supports RAM disk for high-speed 1040ST operation
- Capacities: max. 2 billion characters per file
  max. 64,000 records per file
  max. 64,000 characters per record
  max. fields: limited only by record size
  max. 32,000 text characters per field
  max. 20 index fields per file
- Index precision: 3 to 20 characters
- Numeric precision: to 15 digits
- Numeric range $\pm 10^{-308}$ ti $\pm 10^{308}$

Selected Abacus Products for the ATARI® 人ST™

TextPro

Wordprocessing package for the Atari ST

"TextPro seems to be well thought out, easy, flexible anf fast. The program makes excellent use of the GEM interface and provides lots of small enhancements to make your work go more easily... if you have an ST and haven't moved up to a GEM word processor, pick up this one and become a text pro."

—John Kintz
ANTIC

"TextPro is the best wordprocessor available for the ST"
—Randy McSorley
Pacus Report

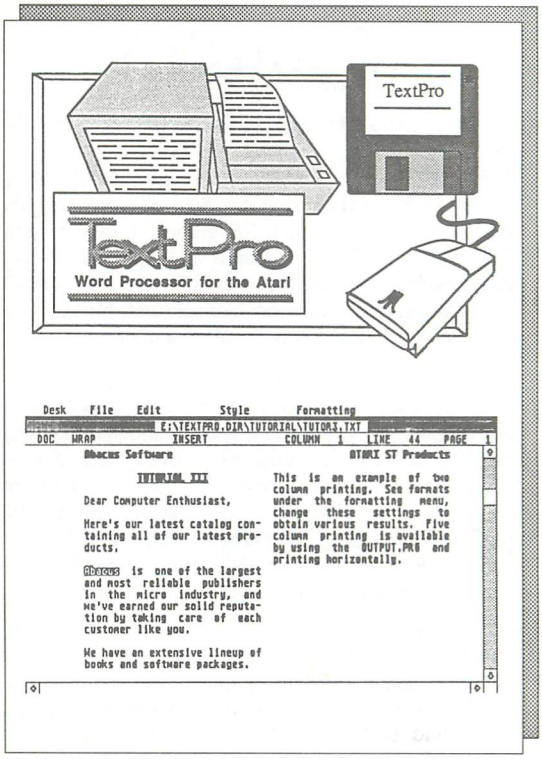
TextPro is a first-class word processor for the Atari ST that boasts dozens of features for the writer. It was designed by three writers to incorporate features that they wanted in a wordprocessor—the result is a superior package that suits the needs of all ST owners.

TextPro combines its "extra" features with easy operation, flexibility, and speed—but at a very reasonable price. The two-fingered typist will find **TextPro** to be a friendly, user-oriented program, with all the capabilities needed for fine writing and good-looking printouts. **Textpro** offers full-screen editing with mouse or keyboard shortcuts, as well as high-speed input, scrolling and editing. **TextPro** includes a number of easy to use formatting commands, fast and practical cursor positioning and multiple text styles.

Two of **TextPro**'s advanced features are automatic table of contents generation and index generation —capabilities usually found only on wordprocessing packages costing hundreds of dollars. **TextPro** can also print text horizontally (normal typewriter mode) or vertically (sideways). For that professional newsletter look, **TextPro** can print the text in columns—up to six columns per page in sideways mode.

The user can write form letters using the convenient Mail Merge option. **TextPro** also supports GEM-oriented fonts and type styles—text can be **bold**, underlined, *italic*, superscript, outlined, etc., and in a number of point sizes. **TextPro** even has advanced features for the programmer for development with its Non-document and C-sourcecode modes.

TextPro Suggested Retail Price: **$49.95**



TextPro ST Features:

- Full screen editing with either mouse or keyboard
- Automatic index generation
- Automatic table of contents generation
- Up to 30 user-defined function keys, max. 160 characters per key
- Lines up to 180 characters using horizontal scrolling
- Automatic hyphenation
- Automatic wordwrap
- Variable number of tab stops
- Multiple-column output (maximum 5 columns)
- Sideways printing on Epson FX and compatibles
- Performs mail merge and document chaining
- Flexible and adaptable printer driver
- Supports RS-232 file transfer (computer-to-computer transfer possible)
- Detailed 65+ page manual

TextPro works with Atari ST systems with one or more single- or double-sided disk drives. Works with either monochrome or color ST monitors.

TexPro allows for flexible printer configurations with most popular dot-matrix printers.

Selected Abacus Products for the ATARI® ST™

PaintPro

Design and graphics software for the ST

PaintPro is a very friendly and very powerful package for drawing and design on the Atari ST computers that has many features other ST graphic programs don't have. Based on GEM™, **PaintPro** supports up to three active windows in all three resolutions—up to 640x400 or 640x800 (full page) on monochrome monitor, and 320 x 200 or 320 x 400 on a color monitor.

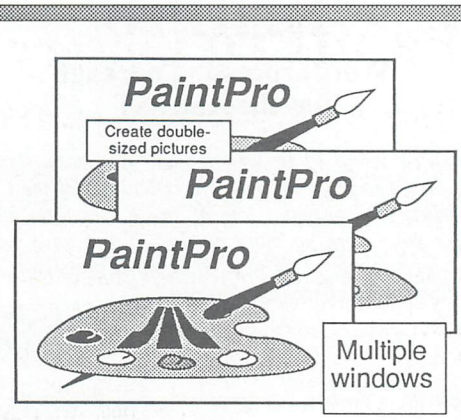
PaintPro's complete toolkit of functions includes text, fonts, brushes, spraypaint, pattern fills, boxes, circles and ellipses, copy, paste and zoom and others. Text can be typed in one of four directions—even upside down—and in one of six GEM fonts and eight sizes. **PaintPro** can even load pictures from "foreign" formats (ST LOGO, DEGAS, Neochrome and Doodle) for enhancement using **PaintPro**'s double-sized picture format. Hardcopy can be sent to most popular dot-matrix printers.

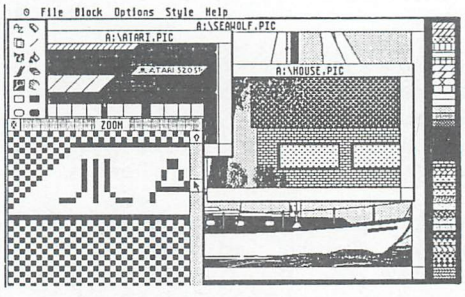
PaintPro Features :

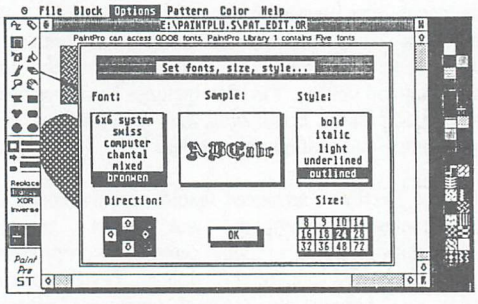
- Works in all 3 resolutions (mono, low and medium)
- Four character modes (replace, transparent, inverse XOR)
- Four line thicknesses and user-definable line pattern
- Uses all standard ST fill patterns and user definable fill patterns
- Max. three windows (dependng on available memory)
- Resolution to 640 x400 or 640x800 pixels (mono version only)
- Up to six GDOS type fonts, in 8-, 9-, 10-, 14-, 16-, 18-, 24- and 36-point sizes
- Text can be printed in four directions
- Handles other GDOS compatible fonts, such as those in **PaintPro Library # 1**
- Blocks can be cut and pasted; mirrored horizontally and vertically; marked, saved in LOGO format, and recalled in LOGO
- Accepts **ST LOGO, DEGAS, Doodle & Neochrome** graphics
- Features help menus, full-screen display, and UNDO using the right mouse button
- Most dot-matrix printers can be easily adapted

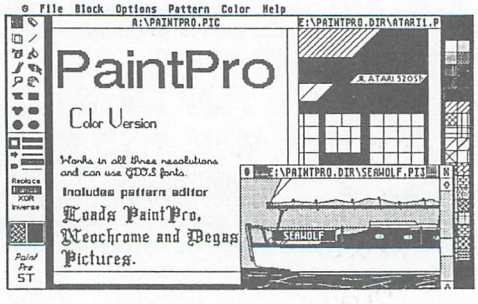
PaintPro works with Atari ST systems with one or more single- or double-sided disk drives. Works with either monochrome or color ST monitors. Printer optional.

PaintPro Suggested Retail Price: **$49.95**









Selected Abacus Products for the ATARI® ST™

PaintPro Library #1

Fonts and Clipart for the Atari ST

The ST's excellent graphics capability make it a natural for computer art and design. To add even more flexibility and features to PaintPro we've released **PaintPro Library #1**, a companion graphics package that contains a diverse range of fonts and symbols for almost every application. It contains five new original fonts for the ST: Swiss, Computer, Chantal, Mixed and Thames. Paint Pro Library #1 also contains scores of new symbols, borders and ornamental lines. As you can see from the examples in the next column, this program fills a real need for your customers' design requirements.

PaintPro Library #1 contains five new specially designed fonts:

- Swiss
- Computer
- Chantal
- Mixed
- Thames (Old English)

Computer

Swiss

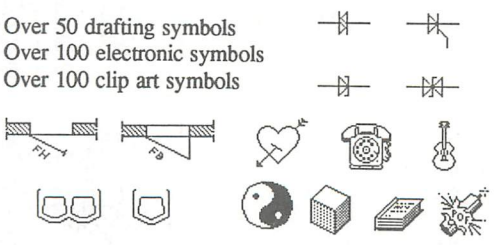
Chantal, Mixed 48

Mixed 16, Mixed 18, Mixed 24

Thames 24 and 48

Also included in **PaintPro Library #1**:

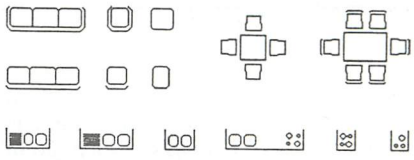
- Over 50 drafting symbols
- Over 100 electronic symbols
- Over 100 clip art symbols



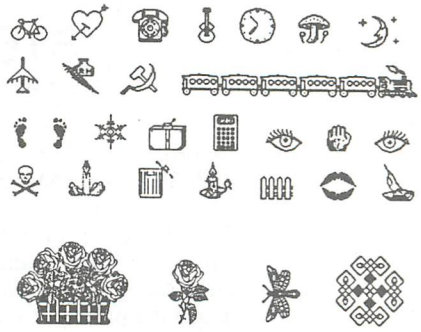
All fonts are GDOS compatible and may be used with "foreign" software that supports the GDOS. **PaintPro Library #1** also has hundreds of symbols, borders, and ornamental lines for use in your graphic designs. These libraries are DEGAS® compatible.

PaintPro Library #1 Suggested retail price: **$29.95**

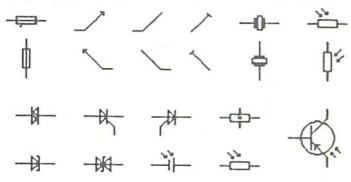
Over 50 drafting symbols



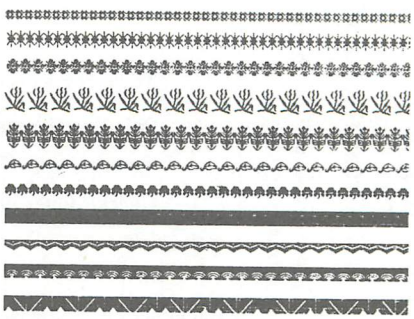
Over 100 clip art symbols



Over 100 electronic symbols



And many decorative borders



How to Order

Abacus P.O. Box 7219, Grand Rapids, MI 49510

All of our ST products—applications and language software, and our acclaimed 14 volume **Atari ST Reference Library**—are available at more than 2000 dealers in the U.S. and Canada. To find out the location of the Abacus dealer nearest to you, call:

(616) 241-5510

8:30 am-8:00 pm Eastern Standard Time

Or order from Abacus directly by phone with your credit card. We accept Mastercard, Visa and American Express.

Every one of our software packages is backed by the **Abacus 30-Day Guarantee**—if for any reason you're not satisfied by the software purchased directly from us, simply return the product for a full refund of the purchase price.

Order Blank

Send your completed order blank to:

Abacus
P.O. Box 7219
Grand Rapids, MI 49510

Your order will be shipped within 24 hours of our receiving it

Name: _____

Address: _____

City _____ State _____ Zip _____ Country _____

Phone: ___ / _____

Qty	Name of product	Price
	Mich. residents add 4% sales tax	
	Shipping/Handling charge	4.00
	(Foreign Orders $12 per Item)	
	Check/Money order TOTAL enclosed	

Credit Card# _____

Expiration date _____ Signature _____

For extra-fast 24-hour shipment service, order by phone with your credit card

The Atari® ST for Beginners

The Atari ST is a "user-friendly" computer. With its icon-based operating system, the ST is ideal for a beginner, although many aspects of the ST can be confusing to the first-time computer user. **The Atari ST for Beginners** will help you learn the essentials of the Atari ST without problems. Some of the topics in this informative book include:

- Setup and connection
- TOS, GEM and application programs
- Introductions to BASIC and LOGO programming
- BASIC program listings with detailed explanations
- Glossary of computer terms
- A short history of computer science
- What else you can do with your Atari ST—now and in the future

About the authors:
Rainer Lüers is a noted teacher of computer science and author. Michael Stein, one of the chief editors of the West German magazine Data Welt, has applied his computer knowledge and his humorous and informal writing style to this book, making the ST easy for everyone to understand.

ISBN 0-916439-55-0

Part of the continuing series of informative books from

you can count on Abacus

A Data Becker Book