

DISK DRIVE GUIDE

688
M
.95



for the Commodore 64™

Nancy L. Wilmot



C-64^{TM.}

DISK
DRIVE
GUIDE

By NANCY L. WILMOT

CON-COR

DISCLAIMER: (No material on computers seems to be complete without one of these, so here it is).

In essence, we assume NO responsibility.

If your system blows up in your face, it's not our fault.

If you see "floppies" in your sleep, it's not our fault.

If you go crazy, it's not our fault.

Every attempt was made to correct typos, (usually due to hieroglyphics) that would cause "technical difficulties." We wish we could guarantee perfection in this area but since we're human, we can't. We hope you'll forgive us for thoooose English typos that were allowed to slide by.

Please, feel free to write us regarding errata, suggestions and comments.

COPYRIGHT — 1984, By CON-COR INTERNATIONAL and NANCY L. WILMOT

All rights reserved. Neither text nor software may be duplicated by any means without written permission. The original purchaser is granted permission to make backup copies of the disk and copy forms from the text as designated for personal use only, not for distribution.

CON-COR INTERNATIONAL
1025 Industrial Drive
Bensenville, Illinois 60106-1297
Attention: Nancy L. Wilmot

ACKNOWLEDGEMENTS

(Alphabetically since the top line isn't long enough)

JAMES M. CONWAY

— started it all by saying, "You ought to write a book."

PATRICIA R. CONWAY

— artistic consultant, inspiration, friend

"PACE MICRO SOFTWARE CENTERS" – Wood Dale & Chicago, Illinois

— time on the SX-64; a "candy store" for computer users

"KINDLY KOMPUTERS" – Lake Forest, Illinois

— time on the dual drive, a classic example of a "dealer" in the best sense of the word

CHARLES OGLESBY, "PAGE ZERO CONCEPTS" – Round Lake, Illinois

— teacher, advisor, hardware man, "down home dealer," more help than I can describe

BILL SCHLAK

— the first working relative file program I ever saw

WARREN TOWNSHIP HIGH SCHOOL – Gurnee, Illinois

— esp: Esther Keefauver, Dick Richey, Elmer Stone, Larry Callan, Dorothy Michno, Julie Cadieux

LYMAN JOHN WILMOT

— lived with computer clutter and a computer addict

DAN MEAD, JACK JUDSON, MARV JOHNSON and CAROL SLADON

— resource people and inspiration

McKNIGHT ASSOCIATES

— somehow put it together so it looked like I wanted it to

C-64, SX64, VIC-20, 1541, 1525 – Trademarks of Commodore Business Machines Inc.

MSD-SD1, MSD-SD2 – Trademarks of Micro Systems Development Inc.

DEDICATION

NANCY and JACK LAYER

CONTENTS

(The first page of each chapter contains its own directory)

0000	INTRODUCTION	1
1000	LOAD	37
2000	FORMAT A "NEW" DISK	53
3000	SAVE	69
4000	OTHER COMMANDS	81
5000	DATA FILES IN GENERAL	91
6000	SEQUENTIAL FILES	103
7000	RELATIVE FILES	143
8000	RANDOM FILES	193
9000	FLASHING RED LIGHT	207
10000	APPENDICES	217



0000 INTRODUCTION

CHAPTER DIRECTORY

0010	Welcome, etc.
0020	What you need
0030	Assumptions about the reader
0040	Text Objectives
0050	About the text
0060	Where to start
0070	Chapter Notes
0080	"Friendly Floppy"
0090	Program Notes
0100	THE DRIVE
0110	Care of the Drive
0120	Anatomy of the Drive
0130	The Drive Communicators
0140	Drive States
0150	Single vs Dual
0160	Drive Number
0200	DISKS
0210	Anatomy - General Parts
0220	Anatomy - External
0230	Anatomy - Internal
0240	Arithmetic
0250	Floppy Development Stages
0260	The Directory
0270	Details, Details (Track 18)
0280	Other Kinds of Disks
0290	Care of Floppy Disk
0300	SYSTEM SET UP
0400	TO POWER ON
0500	TO INSERT DISK
0600	TO REMOVE DISK
0700	TO POWER OFF

To avoid a common source of confusion:

The letter, O, is used only in words. (OPEN, CLOSE, COPY, LO, etc.)

The number, Zero can look like any of these.

0 0 0 0

Generally speaking, use of the letter, O, is avoided like the plague. If a character is not in a word, it's a ZERO 99.9% of the time. This is the kind of thing that computer-philes and computer-phobes must learn to live with.

0010 *Welcome:*

Allow me to welcome you to the wonderful world of the Commodore® disk drive. Whether you are interested in programming or simply want to be comfortable operating your system, I think I can help you.

0011 *Confessions*

I am neither a computer expert nor a professional technical writer. I am a teacher of mathematics (boo! hiss!) by profession and a cronic text rewriter. When my students didn't get along with assigned text, I simply wrote my own. One of them is still in use as the main text in "Advanced Algebra" and of course is being revised constantly.

My computer "expertise" goes back to a FORTRAN course in 1965. At the time, I didn't like the key punch machine, (remember those cards with the holes that you couldn't bend, fold, staple or mutilate?). High school teaching was much quieter than the list printers, card sorters and so forth.

About four years ago, I met a TRS-80®, Model I™ names "Snuffles" and fell in love. Hours passed unnoticed while Snuffles kept saying SYNTAX ERROR. Each fall "Snuffles" had to go back to school and I had no time to play. Soon I got my hands on a faithful VIC-20™ with Datasette. That's when the fun started. Files on cassette are terribly awkward. I gave up files for a while but in the back of my mind I kept thinking that computers should store information. Time for an upgrade.

Peter, (my faithful C-64™), arrived and we got along fine. Two months later came Flopsey, a 1541 Disk Drive. I carefully followed the set up direction and powered on in the specified order. After a few minutes Peter had a nervous breakdown. Fortunately the warrantee was still in effect. Peter II and Flopsey got along fine. I felt a little left out.

The manual had me convinced that I was the dumbest person around until I figured out once and for all that when, on page 8, they said "green," they really meant "red."

To make a long story short, the last 11 months have been spent figuring out how to get Flopsey to do her stuff. She nearly died from the strain and from not having her heads cleaned.

This book is an outgrowth of that experience. I hope it will spare you the hours of frustration that I spent learning things the hard way.

0012 *PROMISES.*

You have my word that:

All the programs have been thoroughly tested.

I'll tell you if I'm not sure if (or why) something works the way it does. My favorite "weasel" words are "speculate," "assume," but most often "think" and "fear."

You have updated information as of January 21, 1984. Beyond that I cannot promise. I keep learning and so will you.

0013 I cannot guarantee that there are no proofreading errors. We tried very hard not to let them slip by. Let me apologize in advance if any cause you trouble.

0020 What You Need

0021 "The System"

C-64™ Computer
1541™ Disk Drive
TV or Monitor

Standard System:
All descriptions in this text refer to this "system."

SX-C64™

The Portable System:
If you are using the SX-C64™, ignore all remarks pertaining to a "green" light (0130). Your drive power is ON when your system is Powered Up. Set up, power on, power off sections should be replaced by your manual.

C-64™ Computer
MSD-SD1™ Disk Drive
TV or Monitor

Compatible Single Drive System:
Refer to your manual for set up, disk insertion and removal. This drive is a bit noisier, not as pretty, a lot faster than the 1541, but is otherwise the "same."

C-64™ Computer
MSD-SD2™ Dual Disk Drive
TV or Monitor

Compatible Dual Drive System:
If you are so fortunate as to be using this drive, you'll have to do a little more work. (See Appendix).

0022 Supplies:

At least one blank disk.

A bookmark.

0023 "Friendly Floppy"

Disk included with text.

0024 A printer (Optional)

Descriptions refer to a 1525 but you should be able to use just about any printer (with suitable interface). Set up procedures may vary. All print routines are very simple. If your printer can print text, (numbers and letters), it should work. If you have graphics capabilities, enjoy them, but you won't need them here.

0030 Assumptions about the reader:

- 1) You are probably not a computer engineer.
- 2) You are probably not a professional programmer.
- 3) Your disk drive skill fits into one of these categories:
 - a) Maiden - Never touched one
 - b) Novice - Can load programs given directions but not comfortable with much else. You may have used software that address the drive (like a word processor that saves documents).
 - c) Intermediate - Can perform elementary operations (load, save, format a disk) but your drive is still functioning like a high speed tape recorder.
 - d) Advanced - Able to write programs for the system that involve all three types of data files. (This text will make a good gift for a friend.)
 - e) Convert - Any of the levels above but with a different system.
- 4) You want to improve your skill level and decrease your frustration level.

0040 Text Objectives:

- 1) Acquaint all users with background information regarding disks and disk drives with special attention to the 1541.
- 2) Aid maiden and novice users with physically operating their systems.
- 3) Aid maiden and novice users to master the elementary procedures of loading programs, saving programs and "formatting" brand new disks.
- 4) Enable all users to deal with the usual "error" conditions.
- 5) Provide a thorough explanation of command sending procedures.
- 6) Familiarize users with DOS 2.6 commands and those available with the supporting DOS 5.1, the «wedge».
- 7) Provide users, familiar with BASIC, sufficient examples and explanations to enable them to write their own programs using all types of data files.
- 8) Provide experiences with data file usage for users who do not wish to program.

0045 This text does not:

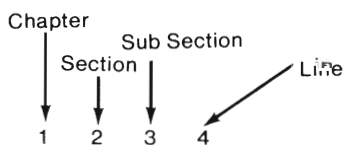
- 1) Intend to teach BASIC programming (procedures for disk drive operations within programs are explained in detail).
- 2) Address using multiple drives. (If you have two drives, you'll be able to adapt).
- 3) Address programming the disk controller.
- 4) Delve into machine language routines.

0050 About the text

0051 The first page of every chapter contains its' own "table of contents" by sub section.

0052 "Line Numbers"

You are now reading line 0052. This is not a sequential text. Users can choose different paths to reach the same goal. The four digit line numbers are used to facilitate cross referencing. Like lines in a program, not all are used. There is no line 0049 in this text. Unlike lines in a program, a single "line" can be a page or two in length.



(1000) refers to an entire chapter.
 (1100) refers to an entire section.
 (1110) refers to an entire sub section.
 (1111) refers to a specific line.

At the top of each page you'll find the range of line numbers on that page.

I suggest you check out each line reference unless you already know its contents. These have been checked but if a reference does not seem to make sense, interchange the middle digits. i.e. instead of 1234, try 1324. If that fails, please accept my apologies. If a reference contains only 3 digits, assume it applies to this chapter, 0000.

0053 Typing:

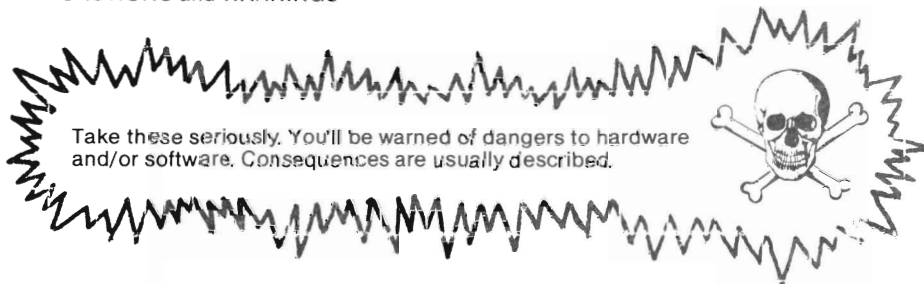
Unless otherwise specified, you are to type exactly what is shown, check for accuracy, and then hit the RETURN key. Substitutions will be noted. If you are not to hit RETURN immediately, that too will be noted. In some cases an extra space can cause command failure. Once you know what should happen, when a command is reproduced exactly, feel free to experiment.

Note: For example purposes only.

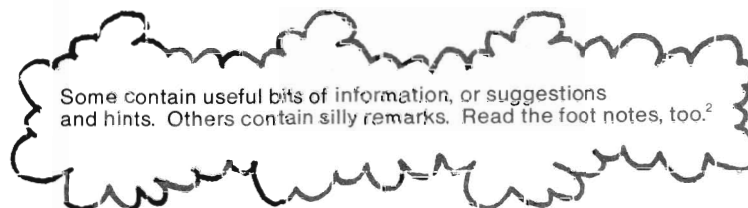
Type: `EXTRA WIDE` Type exactly what has been "printed by the dot matrix printer."

Type: `NORMAL SIZE` By a dot matrix printer.

0054 CAUTIONS and WARNINGS



0055 "Clouds"



0056 BOXES contain general formats.

REQUIRED EXACTLY *variable*

BOLD type - all symbols, including punctuation, are required. *Italic type* - variables and parameters.

0057 `NORMAL SIZE` "printer print" labels show specific examples and titles.

² - Foot notes are kind of like clouds. You can never tell when they are really important. That's life.

0058 Presentation Level

In general, the presentation level varies with the task at hand and the assumed audience. In the early chapters on elementary procedures, I've used a "hand holding" approach. Please don't feel that I'm insulting your intelligence. (If I had a nickel for every hour I've spent overlooking the 'obvious'...) In the later chapters on data files the level changes. By the time you get to those sections, you won't need to have your hand held. Review references are often given if you need them. Some sections or sub-sections are primarily for reference. If all the references were in the appendices where they really belong, we'd have 20% text and 80% appendices.

0059 PIDGIN BASIC

A form of "pidgin-BASIC" is used in the early chapters. It is supposed to provide details for the beginners while allowing the more advanced to safely skip sections. If you've had some programming experience you'll follow right along and possibly become ill at the liberties taken. Read the following definitions anyway. If you have never programmed, don't worry.

"GOTO" Simply means go to the line specified and read on from there. Skip all the stuff in between. It works just like a GOTO statement in BASIC.

"GOSUB" Means go to the line specified but come back when you are finished, (hence the need for a book mark). Your computer gets a RETURN statement to tell it when to go back. With this text, I fear you'll have to figure that out for yourself.

"REM" is computer talk for a remark or comment. Your computer ignores all the stuff after the word REM. You are not a computer so do read the remarks. They often contain tidbits about the section. (if you are a computer that's reading this, call me RIP VAN WINKLE).

0060 Where to start?

Rem: Find the category that best fits your status. Read that line to answer the question.

0061 Maiden Users: Please do not be put off by all the specification in the rest of this chapter. You do not need to master all of that to use your disk drive. Skim (0100) and (0200), but pay very close attention to those on care and maintenance. (0110) and (0290). Sections (0300) - (0700) will explain how to physically operate your system. Those, together with your user's manuals, should get you safely up and running. Go right on to the LOAD chapter (1000). When you've loaded your first program (1220), pat yourself on the head and graduate to the novice status.

0062 Novice Users: Use the first two sections (0100) and (0200) as described above (0061). Check out the rest of the chapter. You may want to adjust some of your operating techniques. The section on inserting disks, (0500), is not as trivial as it sounds. You can probably skip to (1230) in the load chapter (1000), but a quick glance wouldn't hurt. Go right on and master the other elementary procedures. When you've completed (2000) and (3000) you're an intermediate.

0063 Intermediate Users: You are particularly difficult to direct. My best advice is to examine the next section on chapter notes, (0070), along with the individual chapter directories. Be sure to check out the maintenance sections, (0110) and (0290).

If you are not in the habit of using your «wedge», do so. In file work and program development, you'll find it a real time and frustration saver. If you are at all lazy, you'll love it.

The format chapter (2000) not only explains formatting a disk, which you already know, but also provides a background in command sending procedures. It's more than just a "how to."

0064 Advanced Users: Examine the contents to make sure they are suitable. Run a few programs for the heckuvit and please check out (6400). Maybe you can help. Don't forget to gift wrap before you spill something on it.

0065 Converts: Welcome. There are some command summaries in the appendix, at the end of early chapters and in the first section of each of the data file chapters. Find your level above and enjoy.

0070 Chapter Notes:

Rem: Sorry that these aren't right with the chapter. This section is one of my "after thoughts." If I rip this text apart one more time, the publishers will have heart failure.

0071 (0000) Introduction

(Please read preceding sub sections if you haven't already done so).

(0100) The Drive:

(0110) on maintenance is a must for all users. Other sections are background and for later reference.

(0200) DISKS

(0290) on care is a must for all users. Other sections are background and for later reference. I do not guarantee the contents of (0270).

(0300) - (0700) Physical Operations

Primarily intended for beginners.

0072 ELEMENTARY PROCEDURES

(1000) LOAD

(1100) Preliminaries - primarily for beginners.

(1200) Procedures - primarily for beginners but all users should deal with the «wedge» (1230 - 1250)

(1300) Slick Tricks - some fun but can safely be postponed.

(1400) General Formats - may be sufficient for converts. Good for beginners learning to read manuals.

(2000) "Format"

(2100) Preliminaries - good background for all users.

(2200) Procedures - (2210) is sufficient to get the job done. (2220) and (2230) for the curious and/or lazy.

(2300) Analysis and Experiments - may be omitted by casual users. (2310) and (2320) are for serious users. (2330) and (2340) are for the serious users and may be safely used by anyone.

(2400) General Formats - as above.

(3000) SAVE

(3100) Preliminaries - good background for all users.

(3200) Procedures—Strictly "how to"

(3300) Related Operations—all users should cover this section.

(3400) General format - similar to other general format sections. Please note recommended formats.

0073 MISCELLANEOUS

(4000) OTHER COMMANDS - primarily a reference section. Not a recommended starting place for beginners.

(9000) FLASHING RED LIGHT - a must for ALL users.

0074 DATA FILES

(5000) DATA FILES in GENERAL

(5100) Types of files - good background for all users.

(5200) Common Procedures - programmers or potential programmers only.

(5300) Commands - programmers or potential programmers only.

(5400) File Data Format - a must for all but the simplest file management programs.

(6000) SEQUENTIAL FILES

(6100) Preliminaries - primarily for programmers.

(6200) Sample Program - casual users can benefit from the sample program to some extent.

(6300) "COMMAND DEMO" - casual users can benefit from running the program. This section is of special interest to programmers. The programming techniques are much different than those used in the "learning programs" in other sections.

(6400) "A Can of Worms" - may best be saved for last. This section is perfectly safe for your system and disks, but could be hazardous to your mental health.

(7000) RELATIVE FILES

(7100) REL FILES in GENERAL - primarily a reference section for programmers.

(7200) "GEN REL" - sample program. Should appeal to all but the most casual user. Not a good starting point for beginners.

(7300) "GEN-REL" - Detailed explanation for programmers only.

(7400) "GEN-REL" Complete Listing - merely a convenience.

(8000) RANDOM FILES

The chapter follows the same pattern as the RELATIVE FILE chapter. It is not nearly as "Tight" as the two chapters that precede it. To be honest, I haven't fooled with RANDOM FILES all that much. Sequential and Relative files have done an admirable job of handling my Grade Book and Tax Records. Maybe you'll see a need for them and pursue the subject in greater depth on your own.

0075 APPENDICES

(10000) APPENDIX DIRECTORY

- (10100) Procedure checklists — not for beginners. May wish to transfer to file cards.
- (10200) Command Summary (DOS 2.6) — a reference preferred command formats only. «Wedge» Command Summary — especially useful for Intermediate users who may still think that the wedge doesn't do much.
- (10300) Dual Drive Patches
- (10400) Forms a batch of forms for record keeping. A few may help come tax time. Others may help you get organized before things get out-of-hand.
- (10500) ID Code-check list to prevent duplication of disk ID codes. Start using this list as soon as you can load a directory and find the ID code in it.
- (10600) Printer Program Listings — for reference. A study of these will show how easily the printer can be integrated into your programs.
- (10700) Symbol list — for reference. If you are/were confused this will show you one of the reasons why you are/were.
- (10800) Bibliography
- (10900) Index

0080 "Friendly Floppy"

Rem: The programs and some sample files are included on this disk. The numbers refer to text lines to come

0081 Suggestions:

- 1) Make a back up copy as soon as you are able (3320). DO NOT use the ID code, NW. (0260)
- 2) Cover the write protect notch (0221)
- 3) Use your practice disk for all write operations. (3000), (6000), (7000) and (8000).
- 4) Do NOT format² "Friendly Floppy" (2000) and (0250).
- 5) Read the Sections on Care of Floppy Disk (0290) and Care of the Drive (0110).

² - If you know what I meant by that I didn't have to say it. If you didn't, don't worry. It's one of the things you'll learn.

0082 Directory

When you learn to "view the directory," (1210) and (1250), this is what you'll see. The meaning of the directory of disks in general is in (0260), (0270) and explored in (6400).

```

0 ████████████████████████████████████████████████████████████████████████████
22 "PROGRAM NAME" PRG
1 "OPEN EXP" PRG
1 "ERROR READ" PRG
4 "1ST SEQ FILE PGM" PRG
4 "SEQ FILE # PGM" PRG
5 "SEQ FILE PRINT" PRG
17 "GEN REL" PRG
20 "GEN REL PRINT" PRG
6 "RANDOM FILES" PRG
6 "RANDOM PRINT" PRG
15 "COMMAND DEMO" PRG
5 "EXP$-PRINT" PRG
4 "GR SAMPLE" REL
1 "TEST 1" SEQ
1 "RND SAMPLE" SEQ
545 BLOCKS FREE.

```



0090 Program Notes

- "PROGRAM NAME" for use with the LOAD chapter (1000) safe for all users, good for beginners.
- "OPEN EXP" for use with (2331). Safe for all users, of interest to the curious.
- "ERROR READ" for use with (9230). Safe for all users, a sample to use in your own programs.
- "1ST SEQ FILE PGM" for use with (6200). Careless use can damage sample files and possibly result in unwanted scratches. Basically a "learning experience," of interest to programmers just starting on data files.
- "SEQ FILE # PGM" for use with (6260). Of interest to programmers, basically a "learning experience." Careless use can damage sample files and possibly result in unwanted scratches.
- "SEQ FILE PRINT" requires printer. Can be substituted for "1st SEQ FILE PGM". Other remarks as above.
- "GEN REL" for use with (7200) and (7300). Careless use can damage sample files. Can be adapted for personal use with minimal knowledge of programming. Suitable 2nd experience with file programming.
- "GEN REL PRINT" requires printer. Can be substituted for "GEN REL." Other remarks as above.
- "RANDOM FILES" for use with (8000). An introduction to random file programming, show combinations of file types, of no use other than a minimal sample program. Read (6150) before using.
- "RANDOM PRINT" requires printer. Can be substituted for "RANDOM FILES." Other remarks as above.
- "COMMAND DEMO" for use with (6300). Of special interest to the programmer just starting on files. A little slicker than its predecessors, safe for all users. Careless use will place some test files in the directory and foul up further runs until those files are removed.
- "EXP\$-PRINT" for use with (6400). Printer optional. With minor adjustments, it can be used without a printer. Of interest to the very curious, not a good starting place for beginners, but safe.
- "GR-SAMPLE" is NOT a program. Sample file for use with "GEN REL."
- "TEST 1" is NOT a program. Sample file for use with "1st SEQ FILE PGM"
- "RND SAMPLE" is NOT a program, sequential file for use with "RANDOM FILES". Key to the random file hiding on the disk.

0100 THE DRIVE

Your Commodore® 1541™ DISK DRIVE is a precision machine controlled by its own on board "computer" and your C-64™ or VIC-20™. Working as a team, your computer and drive can store and retrieve vast amounts of data quickly and accurately.

The machinery spins the disk at a high rate ( 300 RPM) while moving the read-write head across the disk. It has to be able to pinpoint any one of 684 blocks on the disk.

The circuitry tells the machinery where to go, keeps track of where it is, lets it know what to do, and even does some checking to see if the task has been completed correctly.

Each 1541™ comes with 16K ROM and even 2K RAM. It has not only a disk controller for the machinery but also a complete Disk Operating System (DOS²). The DOS allows you to communicate with the drive via your computer. It contains all the commands you need to do everything from loading a program to creating useful data files.

0101 Your drive contains CBM® DOS 2.6 which alone is sufficient to operate your drive. It can read disks prepared on some other Commodore® drives and can both read and write onto disks from a 4040™ dual drive. (0020) for other compatible drives.

0102 Your system cannot use disk prepared on "Brand X" systems for reasons that will become clear in the following section on disks (0200), and in the chapter devoted to formatting disks. (2000)

0103 Included with your drive is a test-demo disk that contains DOS 5.1. It not only saves typing in many of the commands but it also gives very easy access to some useful processes. The phrase "ACTIVATE your WEDGE" occurs throughout this text.

0110 CARE of THE DRIVE

0111 Move the drive around as infrequently as possible. When **transport** is necessary, pack it very **carefully**. The original container is best unless you've invested in a special carrying case.

0112 Maintain a **clean working environment**. Avoid **smoking** near your drive if at all possible. If you must smoke, invest in an air filter or smoke eating ashtray.

0113 **Cover** the drive when NOT in use.

0114 PROTECT FROM OVERHEATING

Do NOT cover vents when power is on. This could cause overheating. If it is very warm in your computer "room," you may wish to set your drive up on a wooden block so the bottom vent is more accessible. Some of the furniture for computers have vented shelves for the drive. Avoid those that enclose the drive when the power is on.

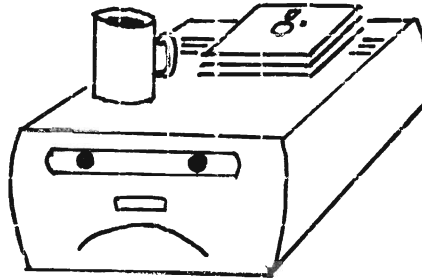
0115 **Clean heads** about once a week. (More often if heavy use or dusty environment). Head cleaning kits are available at your dealer. Do NOT open up and vacuum inside.

0116 If the outer case needs cleaning, do NOT use a spray cleaner in such a way that moisture could get into the drive.

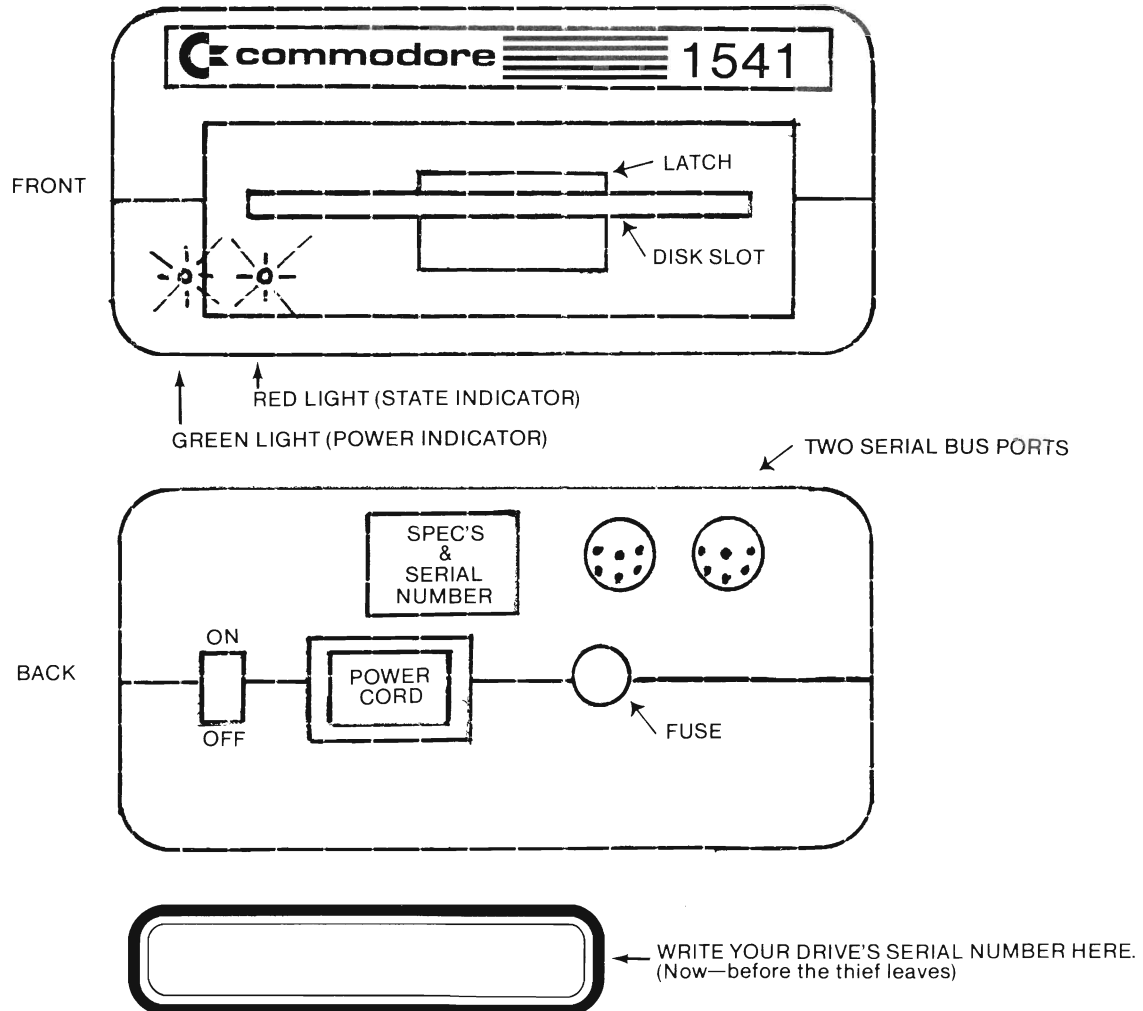
² - "DOS" is pronounced as one word that rhymes with "moss."

® - 1541, 4040, C-64, VIC-20 and CBM are Registered Trade Marks of Commodore Business Machines

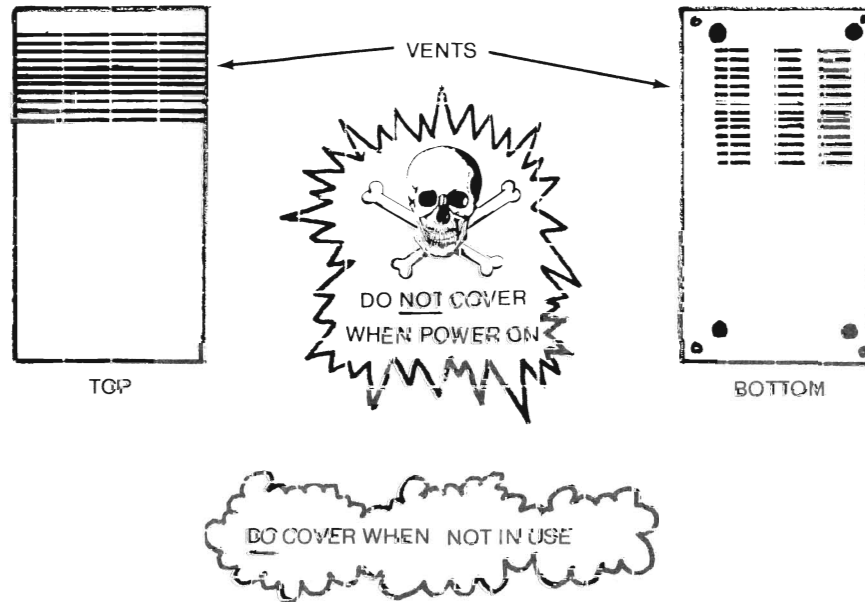
- 0117 Although it is best to avoid eating and/or drinking in your computer room, if you must, at least keep beverages away from the hardware. The top of the drive is NOT a good place for your coffee cup.
- 0118 If your drive does malfunction, have it serviced by properly trained people. See your dealer.
- 0119 Do NOT place disk on drive (0290).



0120 **ANATOMY of THE DRIVE**



0120 (Cont'd)



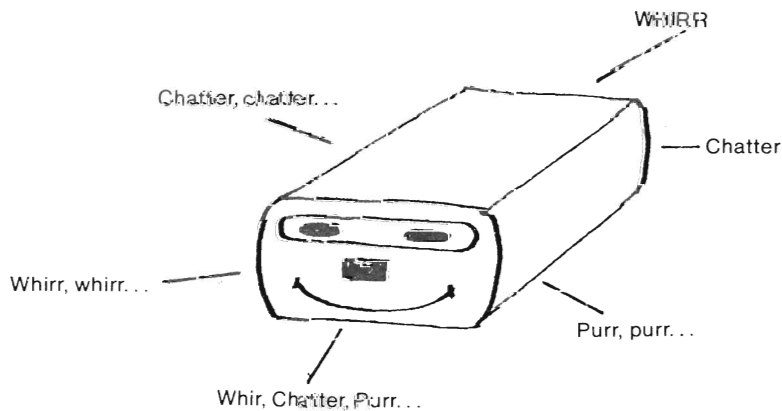
0130 The DRIVE "COMMUNICATORS"²

In addition to using the DOS to communicate with the computer, your drive will "communicate" directly to you via light and noises.

The GREEN LIGHT is the power indicator. It is always on when the drive is on. (i.e. plugged in and turned on).

The RED LIGHT has a larger vocabulary. It can be OFF, ON-SOLID, or FLASHING.

It is perfectly normal for your drive to make noise. A fairly loud "chatter" can be heard as the head moves in search of a track. It "purrs" as it reads and writes. It "whirrs" while it is getting ready to do something. If you've got good ears (or put your ear to the drive) you can detect a "quiet hum" whenever the power is on. If you can't hear it, you can feel it when you put your hand on a "quiet" drive.



² - If your lights do not function as described above, you have a problem. (Occasional flickers, excepted). Return to dealer. These lights are important.

If your drive sounds like a printer for more than a few seconds, return to dealer.

0140 DRIVE "STATES"²

	STATE of DRIVE	GREEN LIGHT	RED LIGHT	"VOICE"
0141	OFF	OFF	OFF	SILENT
0142	WAITING for COMPUTER	ON	ON - SOLID	"Whirr"
0143	READY	ON	OFF	QUIET
0144	WORKING	ON	ON - SOLID (Some flicker)	NOISEY
0145	ERROR CONDITION	ON	FLASHING	QUIET
0146	FILE WORK IN PROGRESS	ON	ON - SOLID	QUIET
0147	SEARCHING FOR TRACK	ON	ON - SOLID (May flicker)	CHATTER

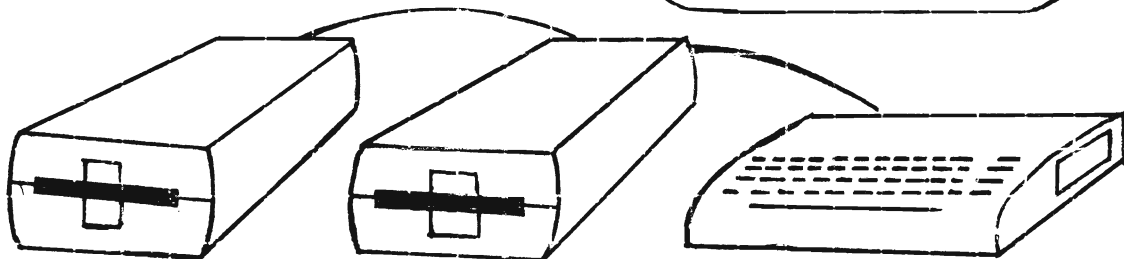
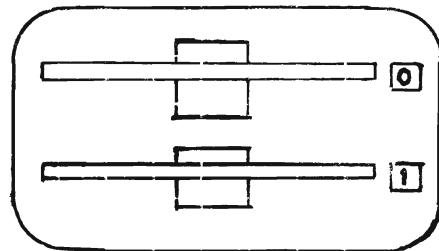
When the drive is READY it is safe to insert or remove a disk. (0500 & 0600) Other times may be hazardous to the health of software and/or hardware.

If your drive sounds like a printer for more than a few seconds, return to dealer.

0150 SINGLE vs DUAL

A DUAL DRIVE system has two drives "in one box." One is DRIVE#0 and the other is DRIVE#1. As you read command formats you'll see lots of 0's. Your 1541™ is a single drive. It is ALWAYS DRIVE#0.

You can use more than one 1541™, but you still have a SINGLE DRIVE system.



DEVICE #9

DEVICE #8

C-64™

TWO DAISY CHAINED SINGLE DRIVES.
(Note - that's DEVICE#, not drive#)

² - If your lights do not function as described above, you have a problem. (Occasional flickers excepted). Return to dealer. These lights are important.

0160 **DEVICE NUMBER**

When you unpack a new 1541™ it is by nature DEVICE #8. It will be #8 until this number is changed via software or by physically altering the hardware.

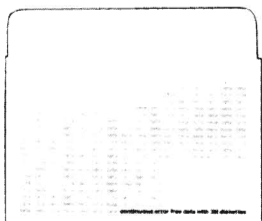
You need not worry about these procedures until you are ready to operate more than one drive. It is best to learn to handle one before you get a second. (If you already bought the second one, leave it in the box until you can drive one drive.)

Each peripheral device has its own number.

0	KEYBOARD	8	DISK DRIVE - unaltered
1	CASSETTE	9	} other DISK DRIVES
2	RS-232/MODEM	10	
3	SCREEN	11	
4	PRINTER (select switch)		
5	PRINTER (select switch)		
6	PRINTER (not for 1525™)		
7	PRINTER (not for 1525™)		

0200 **DISKS**

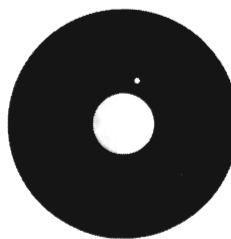
When the words DISK or FLOPPY or DISKETTE are used in this text, each refers to the "single-sided, single density, soft sectored, 5¼" mini-floppy diskette" (whew!) used by your 1541™. These terms are explained in (0280), OTHER KINDS.

0210 **ANATOMY - GENERAL PARTS**

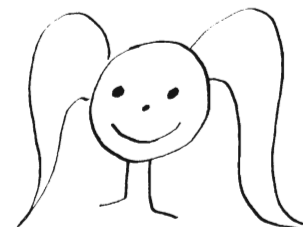
ENVELOPE



JACKET



DISK ITSELF

FLOPPY DISK
(Actual Picture²)

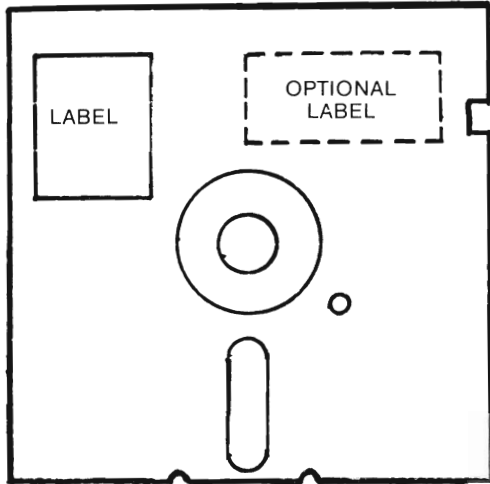
- 0211 ENVELOPE The purpose of the envelope is to protect the disk from dust and abrasion. Do NOT discard.
- 0212 JACKET The jacket enables you to work with a floppy disk. The inner surface is extremely smooth to allow the disk inside to spin at a high rate without being ripped to shreds. Do NOT attempt to open.
- 0213 DISK ITSELF The disk itself is a very thin mylar film coated with a magnetic substance. You can see it through the oval cutout in the jacket. Do NOT touch! Do NOT remove from its' jacket.³
- 0214 FLOPPY DISK² Floppy is a creature that inhabits the pages of this text and takes over when things get a little dry.

² - The picture was sketched during an actual FLOPPY sighting that occurred on the 5th Tuesday of February, 1983. Do you remember the unscheduled full moon that night?

³ - Removal of a FLOPPY from its jacket is tantamount to murder. It is like exposing a vampire to sun light. If you have an uncontrollable urge to dissect a FLOPPY, try to find a dead one. DO NOT dissect expensive software unless you're made of money. Further information is provided to prevent unnecessary FLOPPY-cides.

0220 ANATOMY - EXTERNAL

0221



This is the WRITE PROTECT NOTCH. When covered by tape, (usually supplied by the disk manufacturer), a photoelectric device inside the drive tells it not to write on the disk. It's not real smart to substitute sticky adhesive tape. Seriously, it is a good idea to protect important software from human error. These are usually supplied when you buy a box of disks.

0222 The oval cutout is the READ-WRITE HEAD SLOT. It is through this oval that the drive's HEAD actually contacts the floppy to store or read information. DO NOT TOUCH (even with clean hands). DO protect from abrasion by keeping floppy in the envelope whenever it is not in the drive. Keep all disks not in use in a vertical file. Keep this file some distance from the hardware or any other source of electro-magnetic fields.

0223 The small round hole is the INDEX HOLE. A photo electric device inside the drive "looks" for the hole as a point of reference when looking for a particular sector. If you very carefully rotate the disk in the jacket you will find a hole in the disk itself. (If you find a whole bunch of holes, you've got a 'hard sectored disk.' Take it back before attempting to use. It's not the right kind for this system. (See 0283).

0224 The big round hole is the CENTER HOLE or HUB. You guessed it. It's just like the hole in a phonograph record.

0225 The LABEL is just that. Establish a short code usually the ID code² of the disk. Use only a felt tipped pen to write on this label. Pressure here can damage the floppy and the data it contains.

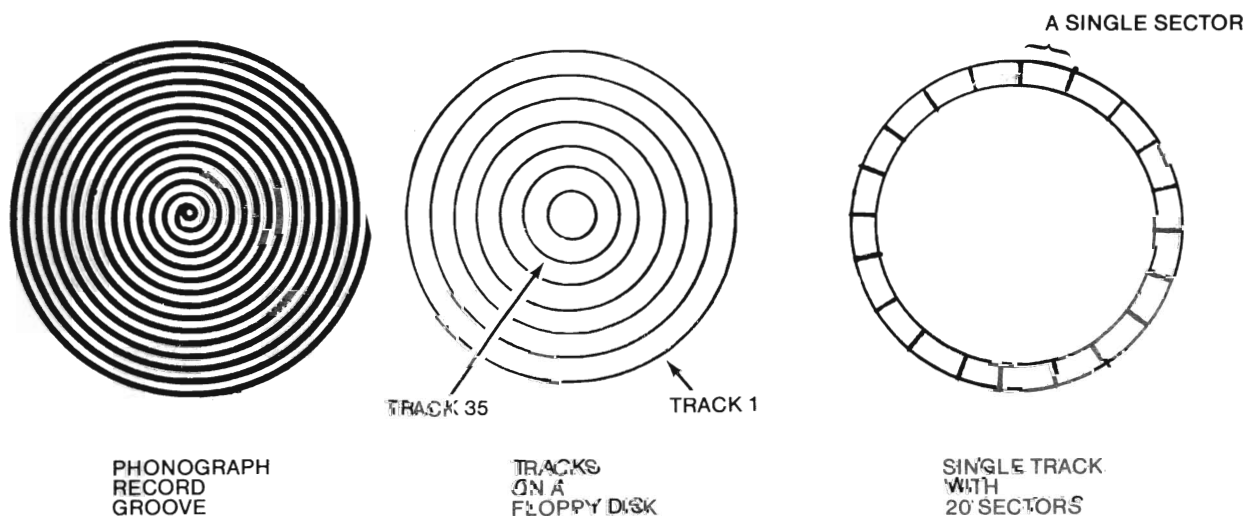
0226 The OPTIONAL LABEL can contain any information you want. Either pre-write (before you put it on the jacket) or use only a felt tipped pen and a very light touch.

0230 ANATOMY - INTERNAL

0231 TRACKS are concentric circles on which information is stored magnetically. These tracks are kind of like the "grooves" on a phonograph record. Standard single density 5 1/4" diskettes have 40 tracks of which our system uses 35. A phonograph record has only one groove that spirals inward. (okay, okay. Two—one on each side).

² - For further explanation see: FORMAT (2000) and DIRECTORY ASSISTANCE (0260).

0231 (Cont'd)



0232 A SECTOR is a small portion of a track. Our system puts more sectors on the outer track than it does on the inner ones. It's a little confusing but it lets our single density disk store almost as much information as some dual density systems. Different systems use different formats. Each system has to "format" a blank disk for its own use.²

0233 TRACK-SECTOR TABLE³ for 1541*

NUMBER OF TRACKS	INDIVIDUAL TRACK NUMBER	NUMBER OF SECTORS ON EACH TRACK	INDIVIDUAL SECTOR NUMBER
17	1, 2, ..., 17	21	0, 1, ..., 20
7	18, 19, ..., 24	19	0, 1, ..., 18
6	25, 26, ..., 30	18	0, 1, ..., 17
5	31, 32, ..., 35	17	0, 1, ..., 16



0234 BLOCK

A BLOCK is sort of a generic term for a sector. (Wonderful—two words for the same thing). To refer to a particular block on the disk we need to specify both a track number and a sector number. Fortunately, unless we are dealing with RANDOM files, the DOS does all the bookkeeping for us.

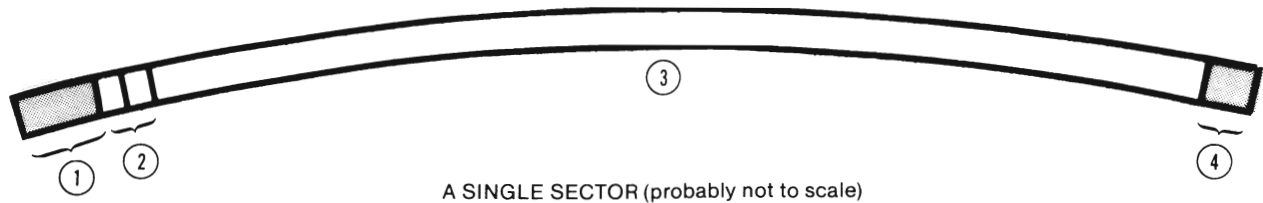
ex. TRACK 20, SECTOR 2 is a BLOCK
 TRACK 20, SECTOR 3 is the next BLOCK
 TRACK 18, SECTOR 0 is another BLOCK

² - Procedures are covered in FORMAT (2000). Don't worry - you don't need to "figure" to do it. The DOS does the work.

³ - Don't bother about memorizing this table. If you're not curious then don't even look at it until you need it.

* - Adapted from "VIC-1541 USER'S MANUAL," CBM, September 1981, Page 55.

0235 ANATOMY OF A BLOCK or SECTOR

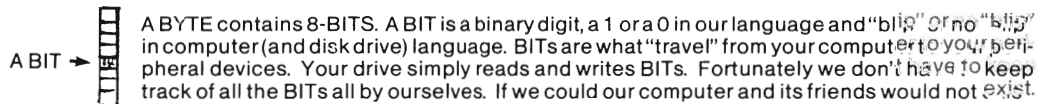


Region 2 contains two bytes used to note the TRACK and SECTOR of the next block in the data file or program.

Region 3 contains 254 bytes for your program or data file.

Regions 1 and 4 are for miscellaneous bookkeeping functions. These enable the drive to know which track and sector it is looking at, check the identification of the disk, allow time to shift "gears" from read to write or vice versa, even check for errors and know when it is finished the sector.

0236 ANATOMY OF A BYTE



0240 ARITHMETIC

Here's a little quiz. Don't panic, the solutions are on the next page. I just figured you'd be sick of reading by now. Time to do some thinking. We'll be working on relating track and sector numbers.

- 0241 QUESTION 1 How many blocks or sectors are there on each of our DISKS? (Hint: See Table 0233).
- 0242 QUESTION 2 If each BLOCK contains 256 bytes, how many bytes can be stored on an entire disk? (Yes, I know you need the answer to Question # 1 to answer this one. Partial credit will be given if you can simply figure out how to figure it out!).
- 0243 QUESTION 3 How many bits are stored on a single disk? (GOOD GRIEF!!)
- 0244 QUESTIONS 4 - 6 There is something wrong with each of the following. Figure out what it is. (Hint: See Table (0233).
 - # 4. TRACK 95, SECTOR 1
 - # 5. TRACK 20, SECOTR 95
 - # 6. TRACK 20, SECTOR 19

0245 QUIZ ANSWERS.

#1. 683 BLOCKS in all.

FORMULA:

$$\left(\begin{array}{c} \text{NUMBER} \\ \text{OF} \\ \text{TRACKS} \end{array} \right) * \left(\begin{array}{c} \text{NUMBER} \\ \text{OF} \\ \text{SECTORS} \\ \text{PER} \\ \text{TRACK} \end{array} \right) = \text{NUMBER OF BLOCKS} \\ \text{ON THAT} \\ \text{BATCH OF TRACKS}$$

WORK:

$$\begin{aligned} 17 * 21 &= 357 \\ 7 * 19 &= 133 \\ 6 * 18 &= 108 \\ 5 * 17 &= 85 \end{aligned}$$

$$357 + 133 + 108 + 85 = 683$$

#2 683 blocks * 256 bytes/block = 174,848 bytes

#3 174,848 bytes * 8 bits/byte = 1,398,784 bits

Who
Cares?

#4 There ain't no Track #95. (We have only 35 Tracks so there isn't even a Track #36.

2 points off
for using
"ain't no"

#5 Sector #95, the 95 is too large. 21 is the largest number of sectors on any one track.

#6 To answer this you have to count like a computer (or at least read the last column, 2nd row of the table).

Sector #0 is the 1st

#1 is the 2nd

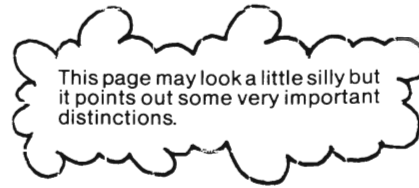
#2 is the 3rd

Etc., etc.

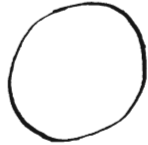
#17 is the 18th

SECTOR #18 is the 19th and last sector on Track 20. Give yourself a gold star if you got this.

0250 **FLOPPY DEVELOPMENT STAGES**



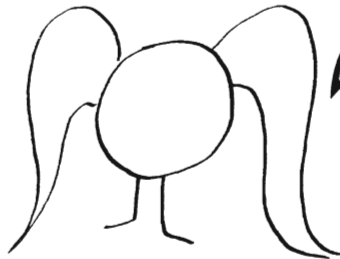
0251 **BLANK**



- Brand New
- No format, No directory, No BAM
- No sectors, No programs, No files.

Note: Any attempt at a LOAD, SAVE or other command, except "NEW" (2000), will result in an error condition — FLASHING RED LIGHT

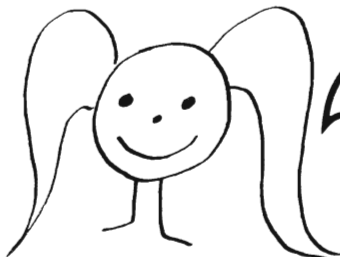
0252 **"BABY"**



I have been "formatted" (2000). I have each of my sectors marked off. Each contains my own ID code. I have a directory and a BAM. I've got 664 blocks free for your programs and files.

Note: You can now LOAD the directory but nothing else. If you ask the drive to LOAD anything else it will flash its red light. You can SAVE programs and write files to your heart's content. Once you save a program or write a file, you've got a "mature floppy."

0253 **"MATURE"**



I carry valuable information for your computer, programs to run and/or files to read. These may have been prepared by a professional or an amateur. In either case a great deal of time and money was spent on me.

Note: If you use the "format process" ("NEW" command) on a "mature floppy" you turn it back into a "baby floppy." (2000)

0260 The DIRECTORY

In essence, the DIRECTORY of a disk is its "Table of Contents." It may have up to 144 entries similar to those shown in the sample below.

0261 SAMPLE DIRECTORIES

"BABY" FLOPPY: 
 664 BLOCKS FREE.


683
 -664

 19

The directory of a "baby floppy" contains no program or file listing. Of the 683 blocks on a disk, 664 are free for your use. The other 19 are used for BAM. (Track 18, Sector 0). The Directory Header, (the heading you see), also on Track 18, Sector 0. Future entries will occupy Track 18, Sectors 1 through 18, (8 entries per sector).

18 * 8
 = 144

"MATURER" FLOPPY

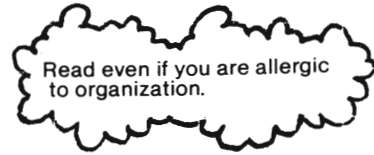
	DISK NAME	ID Code	DOS Version	
				Format type
	4 "REL PGM TEST"		PRG	PROGRAM FILE
	6 "MY REL PGM"		PRG	
	2 "SAMPLE SOUND"		PRG	
	1 "SOUND TEST"		PRG	
	31 "102 GRD BK"		PRG	
	56 "HAUNT"		PRG	
	2 "EXAMINE FILE"		PRG	
	160 "AUTO TEST"		REL	RELATIVE DATA FILE
BLOCK LENGTH OF FILE OR PROGRAM	6 "1000 RECORDS"		PRG	
	3 "MY SEQ PGM"		PRG	
	2 "EDIT TEST"		REL	
	1 "N VS S"		SEQ	
	3 "SEQ PGM II"		PRG	
	1 "TEST CHANGE"		SEQ	SEQUENTIAL DATA FILE
	2 "WIDGET"		REL	
	4 "SEQ FILE CHANGE"		PRG	
	1 "HOW LONG"		SEQ	
	3 "SEQ STRING PGM"		PRG	
	1 "NOW"		SEQ	
	6 "HA DAR CA"		PRG	
	8 "FILE OPEN"		*SEQ	
	199 BLOCKS FREE.			

↑ Number remaining
↑ FILE TYPE²

↑ File that has NOT been CLOSED

² - This is also a USR for USER FILE. Discussion of USR files beyond the scope of this text.

0262 DIRECTORY ASSISTANCE
(or how to get organized)



The sample directory shown is from a 'working' disk. A hodge podge like this results when little or no thought is given to organization. The general idea of a working disk is that one works up a program on it, like a 1st draft (2nd, 3rd, 4th, etc.). When completed (if there is such a thing), the program is transferred to the proper 'using' disk.

The directory header contains, in reverse, the NAME of the Disk, its ID code, a code for DOS version and format type.² The name appears in the header only and is strictly for user convenience. The ID code is written on every sector during the "format process."³

0263 ORGANIZATION BY TYPE

If you anticipate a large file of disks, you may want to set up a system that uses a different disk for each type activity.

ex.

DISK NAME	ID
GAMES 1	G1
FINANCIAL 3	F3
FILE 3	3F
EDUCATION A	EA
GRAPHICS 5	5G

← Notice the difference

← Notice the difference



Each disk should have its own ID code.³ You need not worry about running out of codes. Without resorting to the graphic characters and punctuation, you have 36 symbols (26 letters—10 numerals) at your disposal. Each ID code contains exactly 2 characters. That's over 1,000 possible codes.

$$36 * 36 = 1296$$

1st Slot 2nd Slot

$$1296 - 36 = 1260 \text{ codes.}$$

Because AA is the Same as AA.

0264 Organization by PGR/FILE NAME

Each program/file on a disk must have its own name. A good name is short and descriptive. (Names can never get very long since only 16 characters are allowed).

ex. PROGRAM 1 } Names like these are not good. Two weeks later you won't remember
 PROGRAM 2 } what they are. This kind of name is good for different versions.
 PROGRAM 3 }

 MAILING LIST } Names like these are much better.
 ADD PRACTICE }
 ROCKET }

² - 2A will appear on every disk you 'format' on your 1541 (0261).

³ - The "format process" is described in (2000). The function of the ID code is described in (2123).

0264 (Cont'd)

One frequently has a need to "view the directory" of a disk. The directory of a disk is the only "non program" that can be LOAded into computer memory.² 144 possible entries will take more than one screen to view. (It is like listing anything but a very short program). A neat way to overcome this obstacle is to preface program names with a code to indicate type. You'll then be able to use wild cards and pattern matching³ to view portions of the directory.

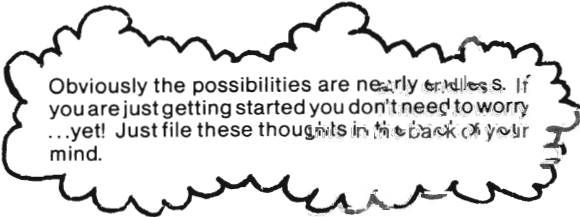
ex. FN - CHECK BOOK
 FN - BUDGET
 FN - TAXES

GM - SNORF
 GM - HAUNT
 GM - GOBBLE

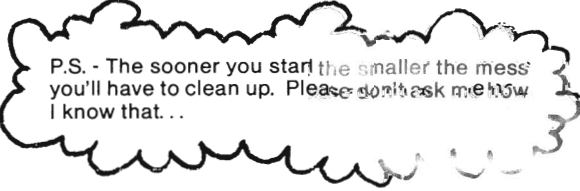
ED - MATH +
 ED - MATH -
 ED - SPELL

SIN - SONG CREATE
 SP - ELEPHANTS

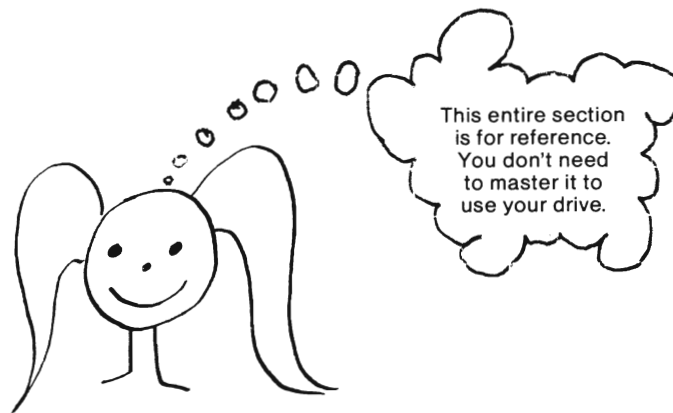
etc., etc.



Obviously the possibilities are nearly endless. If you are just getting started you don't need to worry ... yet! Just file these thoughts in the back of your mind.



P.S. - The sooner you start the smaller the mess you'll have to clean up. Please don't ask me how I know that...

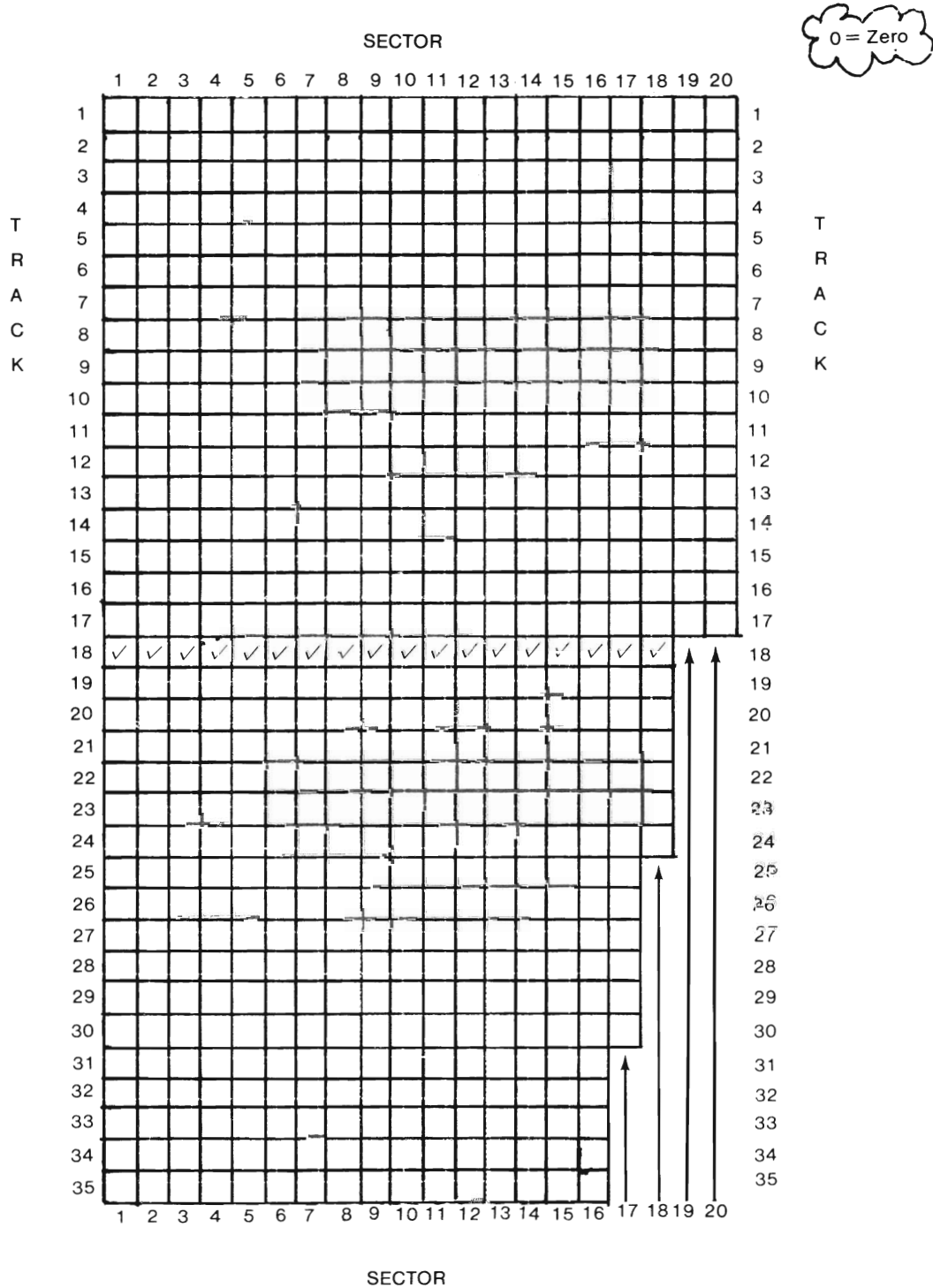
0270 **DETAILS, DETAILS** (Track 18)

² - For methods see (1210). (1250) shows a way to view without overwriting the program surreptitiously in memory.

³ - For details see (1340) and (1350). Especially (3125) for cautions.

0271 Track 18, Sector 0
BAM (Block Availability Map) and Directory Header

When a disk is "formatted" (2000), (0250) and (0233), part of a block is devoted to BAM. Imagine a very small elf sitting on a stool with a tiny clipboard. When a SAVE program or CLOSE file command comes through, the elf checks off the blocks used. The BAM elf can also tell if there are enough blocks available to store whatever it is we want to store, allocate blocks for future use (8000) and free block for future use on a SCRATCH command. Here's how the BAM elf's clipboard may look. Compare with chart (0233).



² - The specific byte numbers reproduced here are adapted from "VIC-1541 USERS MANUAL," CBM, September 1981, Pages 55 - 57. I have reasons for questioning their accuracy. They are presented here simply to give you an idea of "structure." See (6400) for "the problems."

0271 (Cont'd)

Part of Track 18, Sector 0 is also devoted to the directory header.

Directory header (0261)



This is what we always see when we view the directory.

TRACK 18, SECTOR 0 *

- BYTE 0 contains 18 { 0 Track of the first block of the directory
- BYTE 1 contains 1 { 0 Sector of the first block of the directory (0235) region 2
- BYTE 2 contains 65 { 0 CHR\$(65) is an "A." i.e. ASC("A") = 65
- BYTE 3 contains 0 { 0 CHR\$(0) is an empty. ASC("") gets us an error.²

BYTE 4 }
 " }
 " } contain ? { These 140 bytes are BAM. Specifically how they are arranged
 " } ? { I don't know. (Nor did I need to until recently) (6400).
 " } ?
 " } ?
 BYTE 143 }

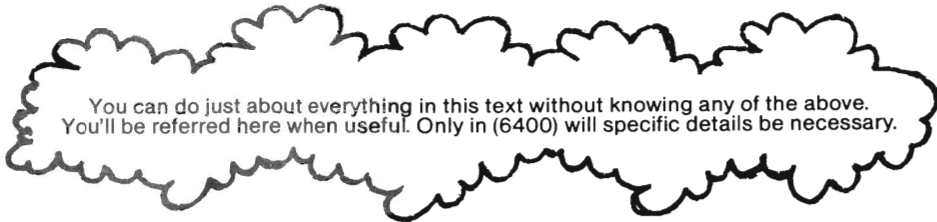
BYTE 144 }
 " } contain { These 18 bytes are reserved for the name of the disk.
 " }
 " }
 " }
 BYTE 161 }

BYTE 161 } contain { ID code for the disk. F & F above, 0 and 2 and N and W in (0261)
 BYTE 162 }

BYTE 165 contains 50 CHR\$(50) is a "2," the DOS³
 BYTE 166 contains 65 CHR\$(65) is an "A," the format type

BYTE 167 }
 ↑ } seem to be used for spacing or not at all. The 1st few are probably
 ↓ } padding of some sort.
 BYTE 255 }

DIRECTORY HEADER



* - Note the foot note on the last page.

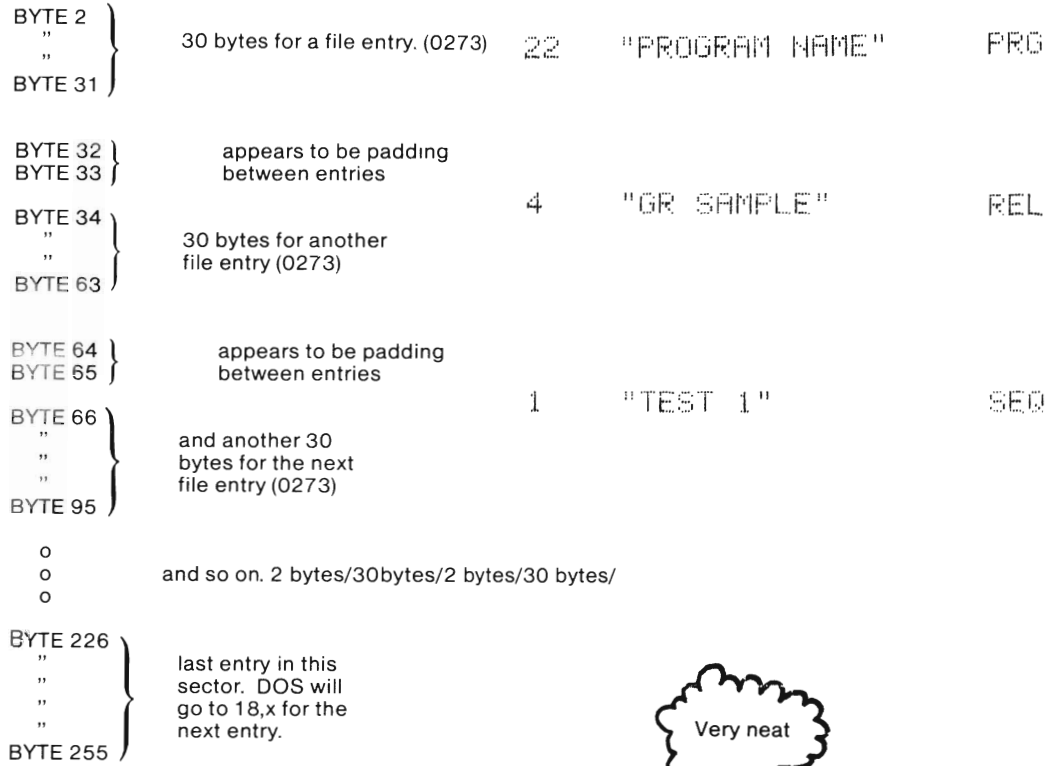
² - See your "Programmer's Reference Guide," CBM 1982 for more details.

³ - I'd like to know where the 6 in DOS 2.6 is hiding of its necessary operation its got to be some place. If not, I'd like to know that, too.

0272 TRACK 18, SECTORS 1 - 18*
Directory Entries

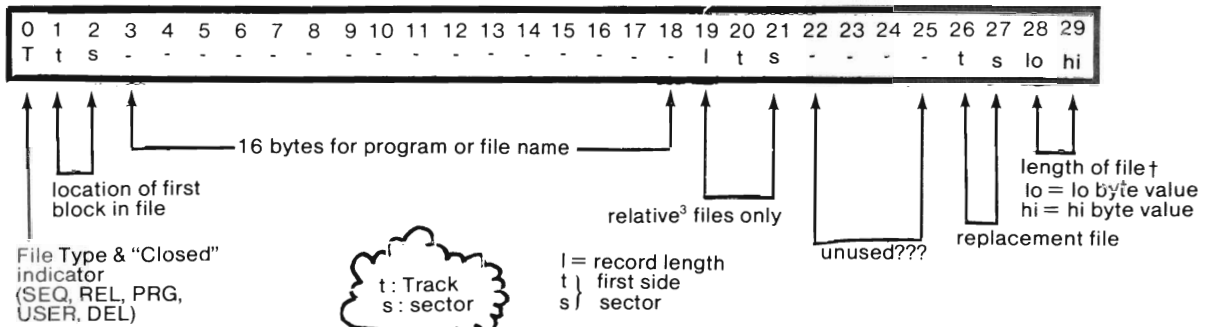
BYTE 0 contains 18 Track of next block in the directory.

BYTE 1 contains 76 Sector of next block in the directory.



Notice that there is room for 8 entries per sector (256 divided by 8 = 32).
Here are 18 sectors on Track 18. 18 sectors * 8 entries per sector = 144 entries in all.

0273 All the directory entries are structured on the basis of 30 bytes.²



* - "VIC-1541 USERS MANUAL" CBM® September 1981, Pages 56 -57

² - It is here that I have my doubts as to the accuracy of the information.

³ - (7000)

† - (7335) shows how these work with record numbers in relative file. lo and hi bytes used in a similar manner in many aspects of computer operation.

0280 **KINDS OF DISKS**

0281 **HARD DISKS** are hard. A hard disk is made of metal (frequently aluminum) impregnated with a magnetic substance. As with a floppy system, information is stored magnetically. In a floppy system, the head contacts the disk producing wear and tear. In a hard disk system the head does not contact the disk (unless it "crashes" but that is another sad story.)

A hard disk system allows for much greater storage capacity, high speed data retrieval and long disk life span. Before you start feeling sorry for yourself, think of the price tag. It contains four figures.

0282 **DIFFERENT KINDS OF FLOPPYS**

SIZES: 8" - Standard
 5¼" - Standard Mini*
 3 - 3½" - Micro Floppy
 Miscellaneous others

0283 **SECTORING:** HARD many index holes, sectors fixed, greater capacity
 *SOFT one index hole, sectors set by system, greater reliability.

0284 **DENSITY:** DOUBLE 70 - 80 Tracks on a Side
 SINGLE 35 - 40 Tracks on a Side

0285 **SIDES:** DOUBLE Both sides contain tracks. Drive mechanism needs "two heads"
 *SINGLE Tracks on one side only. Drive mechanism simpler.

The bottom line(s):

You get what you pay for.

Do you really want to use a bazooka to shoot a chicken?

* - The kind used by your 1541. Double density disks may also be used if you want to spend a little more money. Remember, your system will format the disks (2000), (0230) with 35 tracks.

0290 CARE of a FLOPPY DISK

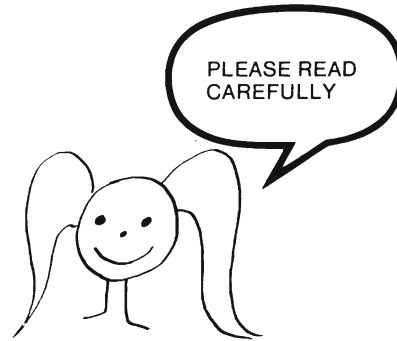
0291 HANDLING

NEVER remove a floppy from its jacket.

NEVER touch anything but the jacket.

NEVER bend a floppy.

NEVER wipe a floppy (even with a soft cloth). If you do see a pet hair, dust particle, cigarette ash, etc., blow gently across the surface. Excuse the indelicacy, but, take care not to spit.



0292 PROTECT from

MAGNETIC FIELDS. Information is stored magnetically and can be erased the same way. Any device plugged into a wall outlet produces magnetic fields. (Get a directional compass and watch the needle bounce as you move it around your system.

TEMPERATURE EXTREMES AND DIRECT SUNLIGHT. Floppies are made of a plastic that will get brittle when too cold and melt when too hot. Your automobile is a lousy place for a floppy on a cold winter night or a hot summer day. When you do transport floppies allow them to "thaw out" before using.

DUST AND ABRASION. Even a tiny dust particle can cause serious abrasions as the disk spins in the drive. Keep a floppy in its envelope whenever it is not in the drive. Write on the label only using a felt tipped pen and a light touch. You can write your life history on the envelope with a hammer and a chisel but remove the floppy first.

MOISTURE. Do not attempt to water your floppy. Floppies only byte bits and can drown in a spill or water ring from a cold drink. Seriously, a spill can ruin software and, worse yet, hardware. Either outlaw beverages from your computer area, or keep at a distance.

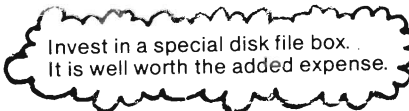
0293 USING FLOPPIES

NEVER POWER ON or OFF when a floppy is in the drive. Data can be erased when the magnetic field changes. Even one messed up byte can ruin a program. Damage does not always result. It's the one time that it does that is one time too many.

ALWAYS insert or remove carefully. NEVER force.

NEVER attempt to remove a floppy when the DRIVE is making NOISE. To be very honest, I don't know what happens. It's an experiment I'm too chicken to try with my hardware, (any volunteers?)^{2*}

ONLY insert or remove floppies when the DRIVE is READY. i.e. GREEN LIGHT ON; RED LIGHT OFF; DRIVE QUIET.



² - If worse come to worst, power down the computer. The drive should stop "working" and you can remove your disk. You will, of course, have to reload the program and you risk losing data entry time.

* - Sources say it doesn't hurt anything, that the heads retract and the drive quiets. Save for a dire emergency when nothing else works. . .

0294 STORAGE

A verticle file that does not put pressure on the sides of the floppies is best.

Do NOT, no matter how tempting, stack floppies on the top of the DRIVE or the MONITER. Again we find magnetic fields and heat.

Keep your storage box a few feet away from your hardware or other electrical devices.



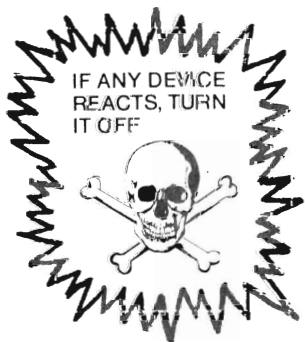
0300 SYSTEM SET UP

0301 This section is intended as a quick refresher. It is especially useful for quick assembly after infrequent transport, or for gremlin checks. (Gremlins occasionally infest computer areas. They frequently take the form of children, parents, siblings, roommates, or spouses).

0302 If you've set up the system more times than you can count, you can skip the entire section. Just remember, it won't work if you don't plug it in. GOTO 0400

0303 If by any chance you are new to your entire system, I suggest you take time out to study your manuals. This section is not intended to replace them for proper unpacking and detailed "set up" instructions. Acquaint yourself with your system bit by bit. GOSUB user's manuals.

0310 SET UP CHECK LIST



	FOR HELP
_____ "Stuff" Gathered	:GOSUB0311
_____ Prep Drive	:GOSUB0312
_____ Prep Printer	:GOSUB0313
_____ Interconnect Devices	:GOSUB0314
_____ Connect Devices to Power	

0311 "Stuff"

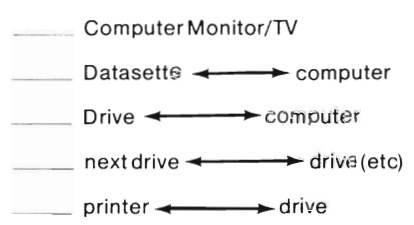
- _____ Computer _____ Disk Drive _____ Printer
- _____ Cable for TV/Monitor _____ Power Cord _____ Ribbon
- _____ Power Cord with Transformer (Black Box) _____ Power cord (probably attached)



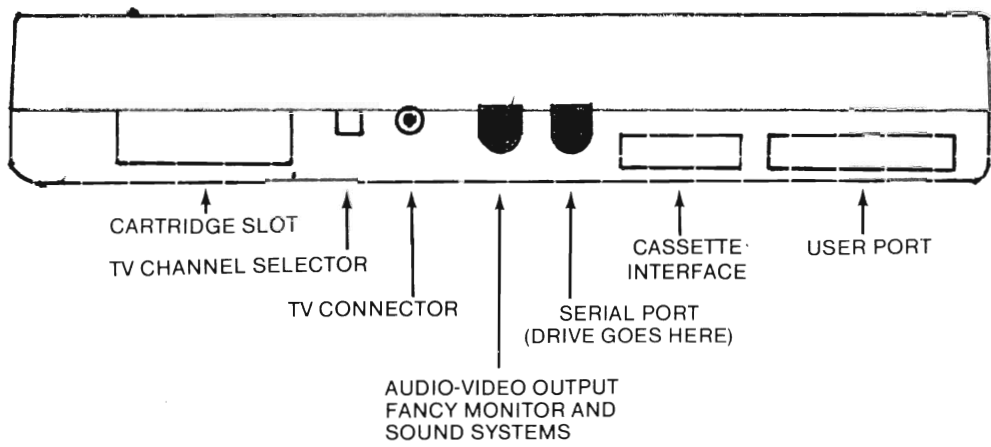
0312 Prep Drive: If shipping guard is visible (in front—you can't miss it), remove or check for disk. GOSUB 0600.

0313 Prep Printer: Load paper (full width to start so as not to cause print head to print over the edge), and ribbon. Study your manuals.

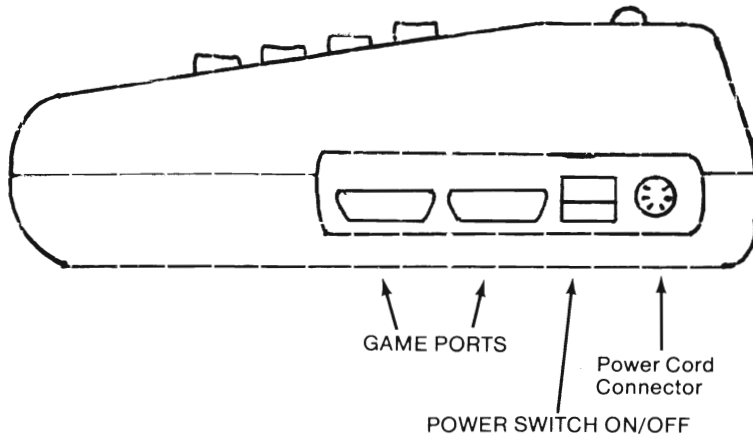
0314 Interconnect Devices



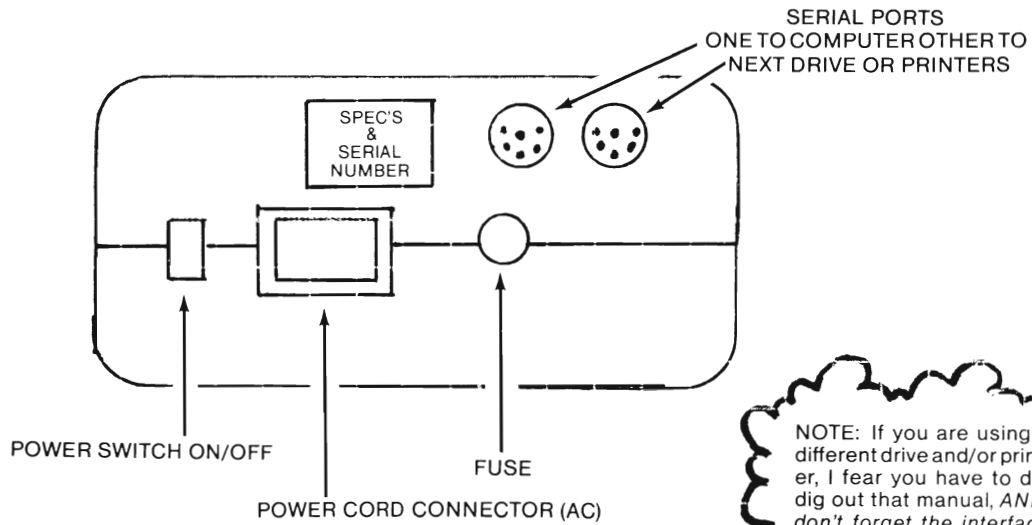
0315 BACK OF COMPUTER



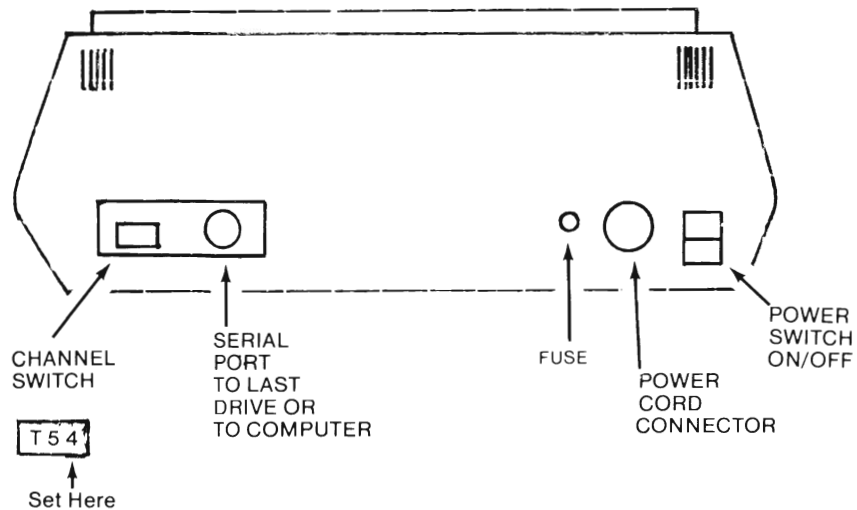
0316 SIDE OF COMPUTER



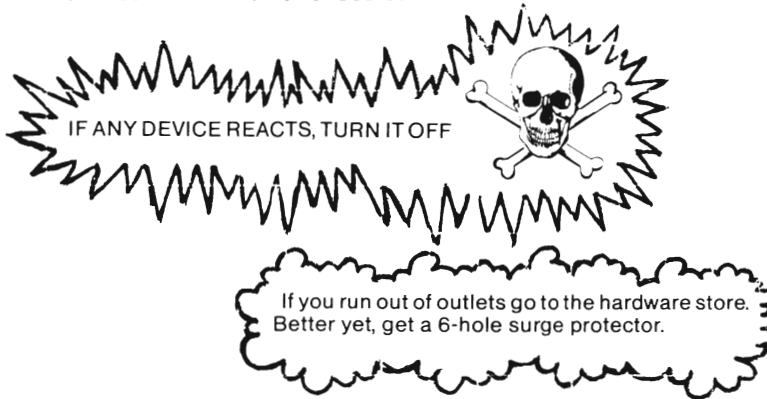
0317 BACK OF DISK DRIVE



0318 BACK OF PRINTER (1525)



0319 Connect Devices to Power Source.



0400 To “POWER ON”

Rem: The order given below and in POWER OFF (0700) is based on often conflicting reports from general sales people, (who may know less than you do), manufacturer’s representatives, self-proclaimed computer experts, (whose knowledge can be worse than none at all²), miscellaneous manuals that contain glaring errors, and technicians who repair equipment. The only consensus is the following—

COMPUTER LAST ON, FIRST OFF

- 0401 Note: If any reaction fails to take place:
- Shut down all devices.
 - Re-check all connections.
 - Repeat procedures from the beginning.

Still no luck? Power down. Re-study manuals and last, but not least, call your dealer.

0410 POWER ON CHECK LIST

	FOR HELP	
_____ SYSTEM SETUP	:	REACTIONS
_____ Cartridges Inserted	:	None
_____ Monitor/TV ON	:	None
_____ Re-Check Drive for Disk	:	Snowy Picture ²
_____ Printer Power On	: GOSUB (0412)	Hopefully none (0600)
_____ Drive Power On	: GOSUB (0412)	Power light on. Print head moves.
_____ Computer Power On	:	Power light on.
		Power light on. Initial image forms.

0411 Drive Reaction: Both lights come on, drive “whirrs.” (0140) The red light may go out and the drive may get quiet when you power up the computer.

0412 Here is the major source of conflict in order. Some sources say drive on before printer.

² - I know of at least one such wig that blew not one but two computers fooling around with wiring tricks.

³ - You may notice your 1541 manual has a little problem with red and green.

* - I use an old TV that needs warm up time. Nobody seems to care when a TV goes on. Maybe a monitor does. Check it out.


0500 TO INSERT DISK

Rem: This section is meant for an inexperienced user, or those wishing to cultivate good habits.

CAUTION: DO NOT insert or remove disk while drive is making noise.

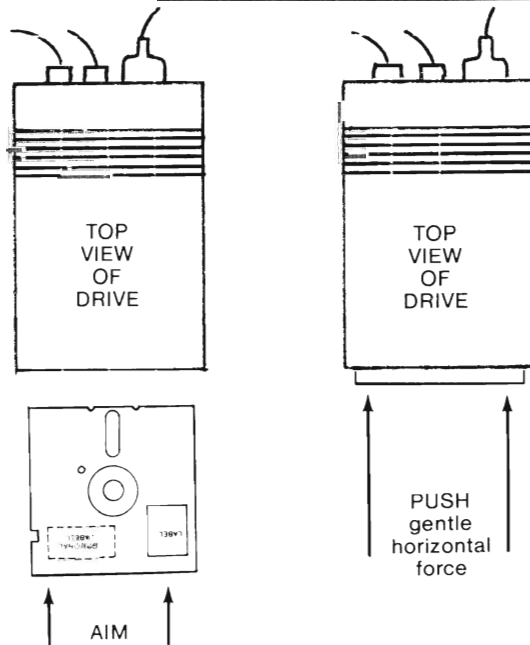
DO NOT power on or power off when there is a disk in the drive.

The major hazard is to the disk. You may get away with other of the above 999 times, it's the 1,000th time that can mess up your software.



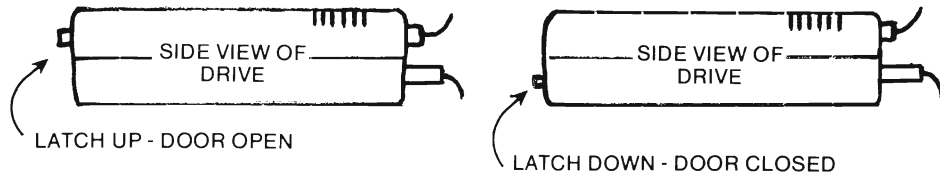
0510 **INSERT DISK CHECK LIST**
(in disgusting detail)

_____ SYSTEM ON	FOR HELP
_____ DRIVE READY	:GOSUB(0511)
_____ GREEN LIGHT ON	:
_____ RED LIGHT OFF	:GOSUB(0130)
_____ DRIVE QUIET	:and
_____ OPEN DOOR	:GOSUB(0140)
_____ GET DISK FROM STORAGE	:GOSUB(0512)
_____ REMOVE DISK FROM ENVELOPE (Not jacket)	:GOSUB(0290)
_____ AIM (see illustration)	:GOSUB(0210)
_____ INSERT (Gently!—see illustration)	:
_____ CLOSE DOOR	:GOSUB(0515)



0511 You can, of course, practice with the system off, just don't turn it on with a disk in the drive.

0512 If you can push down on the latch, the door is already open. If not, push in and allow latch to pop up. If a disk comes flying out, you've mastered most of (0600).



0600 **TO REMOVE A DISK**

Rem: This section is meant for an inexperienced user or those wishing to cultivate good habits.

0610 **REMOVE DISK - Check List**

- | | |
|-------------------------------------------------|--------|
| <input type="checkbox"/> DRIVE READY | |
| <input type="checkbox"/> GREEN LIGHT ON | (0130) |
| <input type="checkbox"/> RED LIGHT OFF | & |
| <input type="checkbox"/> DRIVE QUIET | (0140) |
| <input type="checkbox"/> OPEN DOOR | (0512) |
| <input type="checkbox"/> CATCH DISK | |
| <input type="checkbox"/> PLACE DISK IN ENVELOPE | (0290) |
| <input type="checkbox"/> PLACE IN STORAGE | (0290) |

If no floppy flies out when the latch pops up, and you can feel one inside, wiggle gently. If the write protect notch is covered, a bad job can be the problem. Simply remove and replace. The disk may have a problem. Do not force.

0700 **To POWER OFF**

Rem: Read 0400 if you haven't already done so. The same principles apply here.

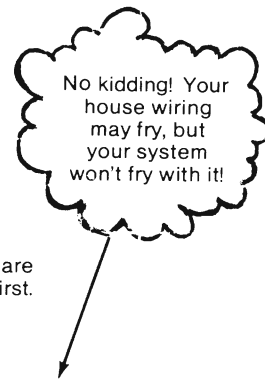
0710 **POWER OFF - Check List**

- | |
|-----------------------------------------------------------------------------|
| <input type="checkbox"/> Check for disk in drive - remove (0610) |
| <input type="checkbox"/> Computer Power Off |
| <input type="checkbox"/> Check for disk one more time |
| <input type="checkbox"/> Drive Power Off |
| <input type="checkbox"/> Printer Power Off |
| <input type="checkbox"/> Monitor/TV Power Off |
| <input type="checkbox"/> Check for power lights |
| <input type="checkbox"/> Unplug system |
| <input type="checkbox"/> Cover |
| <input type="checkbox"/> Thank your system for a good day's (night's) work! |

(picture turns to show)

Nobody seems to care what goes off first.

Just in case lightning strikes.



Depending upon your relationship with your system, you may wish to pat it, or murmur to it in some way. DO NOT burn incense, it is dusty.



1000 LOAD

CHAPTER DIRECTORY

1100 PRELIMINARIES
 1110 What's going to happen
 1120 What you need and how to get it

1200 ESSENTIAL PROCEDURES
 1210 LOAD the DIRECTORY
 1220 LOAD a program
 1230 The WEDGE
 1240 LOAD with Active Wedge
 1250 "View" DIRECTORY
 1260 LOAD to Special Memory

1300 SLICK TRICKS
 1310 Quick LOAD - DOS 2.6
 1320 Quick Load - DOS 5.1 «WEDGE»
 1330 Wild Cards
 1340 Pattern Matching
 1350 Variable Program Names
 1360 Summary Name Find
 1370 Alternatives to Program Loss

1400 GENERAL FORMATS

1010 PRE-LOAD CHECKLIST

CHECK	FOR HELP GO SUB
1011 SYSTEM SET UP	0300
1012 SYSTEM ON	0400
1013 DRIVE READY	0140
1014 DISK SELECTED	1122
1015 DISK INSERTED	0500
1016 PROGRAM NAME HANDY	1123
1017 READY TO SACRIFICE PROGRAM NOW IN COMPUTER MEMORY	1370

CAUTION!!! All LOAD operations replace the program currently in computer memory with the program specified in the LOAD command.



1100 PRELIMINARIES

1110 **What is going to Happen**

REM: See 1200 for "how to" details.

1111 USER types command in proper form and hits RETURN key.

1112 COMPUTER tells DRIVE to go to work and sends a message for you to the monitor. (The screen does not blank.)



SEARCHING For----

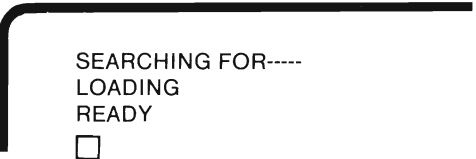
1113 DRIVE puts on its RED WORKING LIGHT and makes noise. (0140) It looks through the directory for the program it was asked to load, notes the Track and Sector of the first block in the program, positions its head to that location and starts to read.

Meanwhile the computer gets the all clear and sends you another message.



SEARCHING FOR-----
LOADING

1114 If the program is longer than one block, the drive will note the track and sector of the next block, read the current block, send that information to the computer, find the next block and so on and so on until it "sees" the end of file marker. It lets the computer know it's finished and the computer sends its favorite message.



SEARCHING FOR-----
LOADING
READY

1115 The red light goes out and the drive is quiet.

1116 All of this will happen very quickly unless the program is very long. (50 blocks take about 30 seconds)

1117 If anything goes wrong, (DOS can't find what it's looking for) the RED LIGHT will FLASH. No need for panic. (yet)

1118 Control of the "FLASHING RED LIGHT" is described in the last chapter 9000. At least locate that chapter, skim it, and remember it's there when you need it.

1120 What You Need and How to Get It

REM: You MUST:

1121 have the System. If no system, GO TO dealer.

1122 have a DISK with a PROGRAM on it.

You have this already since you have the DISK that came with this parcel or the TEST-DEMO disk that came with your system.¹ In general, any previously prepared software "store bought" or otherwise will do if it's for a C-64 or a VIC-20.²

You may not³ perform a LOAD operation on a brand new blank disk or on a disk that has just been formatted. (250)

If you are a first time user I recommend you use "Friendly Floppy" the disk supplied.

1123 know the EXACT⁴ name of the program.

To LOAD a program from a disk, you must tell the system exactly what program to load. Your old tape drive doesn't need the name but you know it has to go from beginning to end. (Back again if you missed it the first time.)

Software documentation (instructions) usually includes the name of the program and any special loading instructions. (No documentation? Don't panic! We'll deal with that in 1130.)

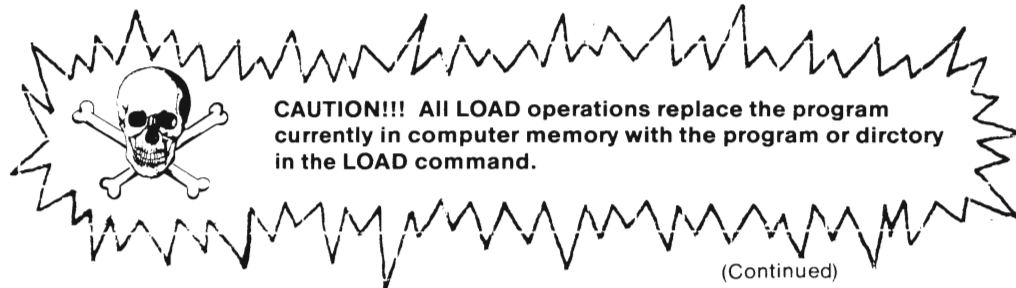
1124 know the correct command format.

You'll find the commands for the disk drive a bit more complicated than those for the tape drive unless you learn to activate the «WEDGE» (1230).

The LOAD commands are explored in disgusting detail in 1200 or in summary form in 1400.

1200 ESSENTIAL PROCEDURES**1210 To LOAD the DIRECTORY**

REM: If this is your first time, I recommend you use "Friendly Floppy" that came with this text.



¹ Be careful with some of these. Mine weren't friendly.

² See also 101-102.

³ You can try, but you'll get a "flashing red light". If you know how to deal with it, no damage to anything but your ego.

⁴ You can get around this if you know some "slick trick". See also 1300.

1210 (Cont'd)

Use $\$$ for the name of the program. ¹

1212 Perform pre load check (1010)

1213 type: ² LOAD" $\$$ ",S

1214 After you hit "return" the system will take over and react as described in 1110.

1215 If the red light flashes, GO SUB 9000.

1216 When the computer says "ready" and the red light goes out type: LIST ³

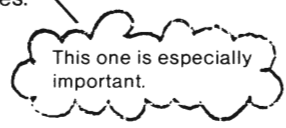
1217 After you hit RETURN key you'll see the directory of whatever disk is in the drive. (See 261 for sample.) You can remove it from memory like you would a program ("NEW", LOAD another program, but don't power off until after you remove the floppy.)

1218 If the directory is very long, you can control the listing in the same way you control a program listing, CTRL key to slow or RUN/STOP key to stop.

1219 If you were here to get a program name, don't forget to remember it. Simply LIST again if you just cleared the screen.

1220 LOAD a Program (DOS 2.6)

REM: PROGRAM NAME must be the exact name of the program you wish to load. Exact means just that, character by character including all of the spaces.



On your "Friendly Floppy" there is really a program by the name of PROGRAM NAME . If "Friendly Floppy" is in the drive and you type the command exactly as shown, PROGRAM NAME will be loaded.

If you want to load some other program, just substitute its name for PROGRAM NAME .

If the documentation has loading instructions that differ from the one below, follow them. (See also 1260)

1221 Perform PRE LOAD CHECK (1010) using the command below.

1222 type: LOAD"PROGRAM NAME",S

(Continued)

This Number Required

¹ The directory is not really a program. It is a special "file" that can be loaded. (No other data file can be loaded directly to memory.)

² Type exactly what you see after the colon.

³ If you try a RUN you'll get a SYNTAX ERROR. No big deal. No flashing lights.

1222 (Cont'd)

REM: The in the command above is the DEVICE NUMBER. This is how the system knows you want it to load from disk rather than cassette. If you don't use the , the system will automatically try to load from tape. When that happens (as it does to old cassette users in a hurry) simply hit RUN/STOP key and start over. No harm done. (See also 160.)

1223 After you hit RETURN key the system will go to work as described in 1110.

1224 If all went well you're ready to LIST, RUN,* or whatever.

1225 If the red light is flashing, don't panic. The most common problem here is mis-typing the program name. (The system may even have sent you a message.) Check your spelling again. If that's the problem, retype or patch up your command. The red light will take care of itself.

1226 If the red light is still flashing, GO SUB 9000.

1230 The WEDGE

REM: To be honest, if all you want to do is load and run programs, the WEDGE is probably more trouble than it's worth. BUT, if you are going to do some serious programming, especially with files or experiment with disk commands, the WEDGE will more than save the time it takes to load.

1231 "ACTIVATE your WEDGE" means to LOAD and run it according to the instructions given in your documentation. I'm sorry I can't be more specific but different programs require different methods. One of the following will probably be required.

a) Load a simple BASIC program which when RUN will put a machine language program (DOS 5.1) into computer memory.

b) Use a special memory load command allowed by a SYS command (See 1260) to put DOS 5.1 into memory.



In either case, follow the directions carefully and be sure to perform the pre load check. (See 1010.)

Once "activated" your WEDGE will stay active until you power off or type the "QUIT"

command, >Q or @Q. No other BASIC command (to the best of my knowledge) will turn off your WEDGE (actually DOS 5.1). Your system will behave normally except that it will respond to a whole bunch of commands that it didn't know before.

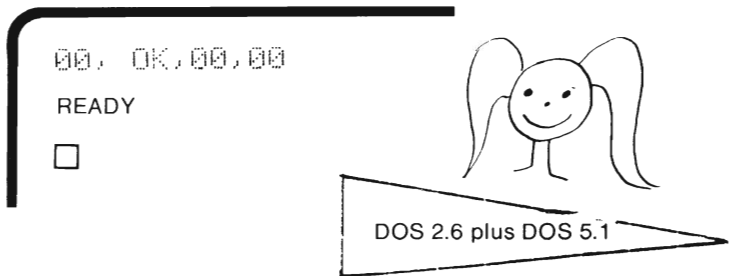
1232 "ACTIVATE your WEDGE" according to directions.

Check for Active WEDGE.

type:  

If your WEDGE is active, you'll see:

1234 P.S. YOU NOW HAVE THE BEST WAY TO SHUT OFF THE FLASHING RED LIGHT.



```
00, OK,00,00
READY
█
```

DOS 2.6 plus DOS 5.1

*Be sure to follow any special instructions given in the documentation.

1260 LOAD to Special Memory Location

REM: This command causes the system to load a program to the exact memory location from which it came. It is only necessary for memory-dependent programs such as those that involve machine language or other esoteric functions.

CAUTION: This command can cause a system "crash" if used when not appropriate.

DO NOT USE unless told to do so, or you know the program was SAVED using the corresponding command or unless you want to see your system crash.*



The only reason for presenting this command at this time is so you aren't bothered if you encounter it in documentation. Since this command can cause trouble, don't type anything unless you know you have to. (See program documentation.)

1261 Perform pre LOAD check. (1010) You may be able to do that mentally by now.

1262 Format: `LOAD "PROGRAM NAME" , S , 1`

REM: It is safe, I hope, to use this with PROGRAM NAME on "Friendly Floppy"

, 1 is the secondary address. It's the signal to the system to load to the memory address from which the program was saved.

REM: For comparison:

`LOAD "PROGRAM NAME" , S , 0` The "formal" format for

`LOAD "PROGRAM NAME" , S` (i.e. If not specified the system assumes a secondary address of 0, which signals a normal LOAD.)

1263 If WEDGE ACTIVE, the following command may be used:

`^PROGRAM NAME` (equivalent to 1262)

*So you want to see your system crash. Use the command in 1262 with "HOW PART TWO" on the "TEST/DEMO DISK" weird things will happen to your screen. REMOVE DISK from drive! Try to get your system to listen. RUN/STOP, RUN/STOP & RESTORE, etc. When you get tired of it, disk out, power off.

1300 SLICK TRICKS

REM: As with the earlier sections, most of these things are easier to do than they are to describe.

The first two sections are for those of you who would rather play with the cursor controls than type.

The 3rd and 4th sections are loads of fun if you like patterns and are good for getting you out of trouble when you want to load a program but don't know its name.


The next section, explaining how to load a program from a program, shows you how to use string variable names for program load operations.





The last two sections aren't really slick, but may be a handy reference.

By now, you should be advanced enough that I can stop boring you silly with pre LOAD checks and lots of other "what ifs". Please use common sense.

1310 Quick Load - DOS 2.6


1311 Put the directory onto the screen. (1210 or 1250)

1312  (cursor up) to the line of the program you wish to load.

-  cursor up
-  cursor left
-  cursor right
-  cursor down

(do not hit return until told to do so.)

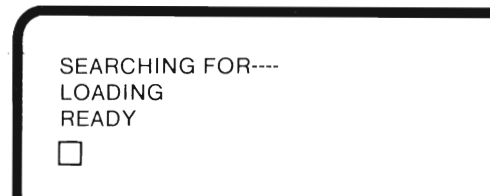
1313 type: `LOAD` in front of the first set of quotes in front of the name.


1314  (cursor right) past the quotes after the program name.

1315 type: `· 8 *` after the second set of quotes.

1316 SPACE BAR to blank out the rest of the line.

1317 Now hit RETURN. You'll see that standard display printed over the rest of the directory. It may look a bit like garbage.



1318 Before you run or list, either type the command and blank the entire line before hitting return or  to a clean line and type the command normally.

Include secondary address `· 1` only if necessary. (1260)

1320 **QUICK LOAD DOS 5.1 «WEDGE»**

We'll use the same process here as in 1310. This way is just a tad quicker. If you haven't done that section yet, do it now to save repetition.

1321 Repeat line 1311.

1322 Repeat line 1312.

(DO NOT HIT RETURN UNTIL TOLD TO DO SO.)

Type: `</code>` in front of the program name. Take care to blank out block numbers in front of the quotes. It's OK to leave the quotes. Remove them if you prefer.

1324 `=>` past the program name.

} optional

1325 Blank out the rest of the line with the SPACE BAR.

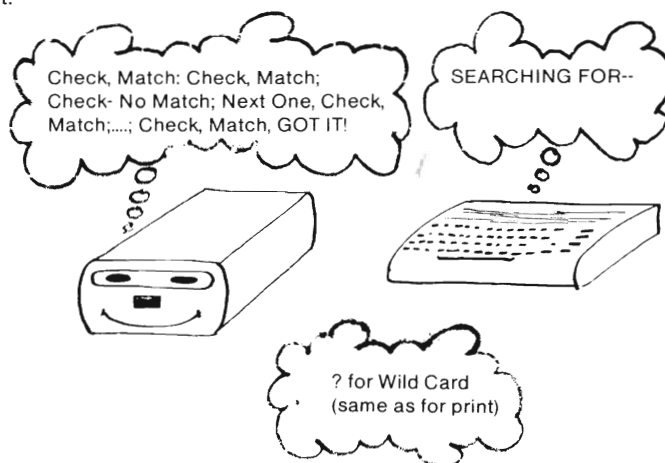
1326 HIT RETURN

1327 Read line 1318 before you run or list.

1330 **WILD CARDS**

1331 Exit program mode. (I'll write like a person for a while.)

When your drive is asked to LOAD a program by name, it checks the directory entries against the name given in the command. It must be able to match each of the 16 slots EXACTLY² character by character. (You get a flashing red light if it can't make a match.



To the drive "PROGRAM 1" does not match "PROGRAM1". If asked to load "PROGRAM-1" it won't load either of the above because it expects to find a "-" after the 1, not a space or a 1. You've got a problem if you can't remember the name. Rather than generating error conditions or viewing the directory, we can use wild cards and pattern matching. (1340) to solve the problem. (Suppose you can't remember whether you called the program TICK or TOCK)

1332 The symbol, ?³, is the wild card symbol.

The command `LOAD" T?CK" , S` will cause the drive to skip its match of the second character. It will load the first program it finds that has for characters (T, first; C, third; and K, fourth) i.e. To the drive T?CK matches all of the following:

TICK, TOCK TACK, TECK, T-CK, T3CK, T CK, in fact, T anything CK

1333 /T?CK is the equivalent WEDGE command. Special memory and corresponding WEDGE commands may also be used. (1260)

² - See 3125 for a description of legal program names.

³ - Yes, the same symbol as the abbreviation for PRINT. The system can tell the difference from the context.

1334 Wild cards can be used to examine portions of the directory.

```
LOAD "$: T?CK", 8
LIST
```

DOS 2.6

“View” all directory entries that match in the 1st, 3rd, and 4th slots.

```
@$: T?CK
```

To “view” with
DOS 5.1 «WEDGE»

Both are useful if you have a long directory or if your directory is organized as described in directory assistance. (264)

1335 Wild Card Formats.

```
LOAD " ", 8
```

DOS 2.6

```
/
```

DOS 5.1
«WEDGE»

Will load the first program found that matches each symbol except those slots filled by ?

```
LOAD "$: "
```

DOS 2.6

and

```
/$:
```

DOS 5.1
«WEDGE»

Will load or view all directory entries that match each symbol except those filled by ?. If no match is found, you'll simply see an empty directory.

```
@$:
```

DOS 5.1
«WEDGE»

to “view” but not load. (1250)

1336 More Samples of Wild Card.

???? matches any name that contains exactly 4 symbols.

?????? matches all names that have exactly 6 symbols.

P?? matches 3 symbol names that start with P.
ex. FIG, PIN, P12, P-2, but not P2, POKE, or PROGRAM 5

????S matches all names with exactly 5 symbols that end in S.
ex. SOCKS, 1234S, C-64S , but not PS or HOPS or BLOCKS

?E? matches all names with exactly 3 symbols that have E as the 2nd symbol.
ex. PEG, EE4, but not PEEK

1337 With a little planning you can set up some nifty directory entries.

1338 With Wild Cards alone, you still have to know the number of letters in the name of the program.

1340 PATTERN MATCHING

1341 The symbol for pattern matching is * .

1342 When * appears in a load command, the drive will only attempt to match characters that precede the * .

1343 Formats for examples:

LOAD " " , 8	or	/
DOS 2.6		DOS 5.1 «WEDGE»

will LOAD the first program found that matches. If no programs have been loaded from the disk in the drive, it will load the first program in the directory that matches.

LOAD "\$: " , 8		/\$:
DOS 2.6		

will LOAD all directroy entries that match up to the as in the examples below.

@\$:

Will show matching directory entries without replacing program in memory.

NOTE:

Examples:

1344

ABC* Loads the first program found that starts with ABC no matter how many symbols follow .

or

Shows all directory entries that start with ABC, no matter how many symbols follow.

*for pattern matching is the same as the "multiply" symbol.

ABC* matches ABC1, ABCDEFGH, ABC---XYZ, but not AABC, or XABC, or -ABC.

↑
blank

* Loads the first program found. It will be the 1st program in the directory if no programs have yet been loaded. It will be the last program "looked at" otherwise.

or

Show the entire directory. i.e. LOAD"\$" , 8 is equivalent to LOAD"\$:*" , 8 so there is no real need to bother.

NOTE:
You might run into an error condition (DISK ID mismatch) when you are fooling around with Wild Cards and Pattern Matching and change disks. The drive will still look for ID codes. Type: @I . If that doesn't work. GO SUB 9300.

1345 Combining Wild Cards & Pattern Matching

You can use both * and ? in the same Load or Directory view operation. (The formats are the same as in 1343.)

1346 Examples Combining * & ?

??T* will match a program name of 3 characters long or longer.*
 ex: BAT MOBILE, CATAclySM, RST, or TTT-123, but not TTXA or IT T



F?* will match a program name 2 characters long or longer* that has F as the first symbol.
 ex: FAT, FINANCIAL, FUN & GAMES, FX but not F or AFTER or FXFFF

A*EC will match all program names of any length* that start with A. The drive will stop looking when it sees the star.

1347 REM: If you happen to be a "word person", the type who knows that OTTO spelled backwards is OTTO and inside out is TOOT, you could really have a lot of fun with these.

1348 REM: Even if you don't want to use these concepts in loading you may find them very useful later.

1349 "Friendly Floppy" contains some programs for you to "fool around with".

Activate your WEDGE. Then "fool around" with these.

Use the format: @ \$:

Try these: E* or ?R* or R* or RA* HAVE FUN!!!
 Examples

1350 VARIABLE PROGRAM NAMES

1351 String variables may be used in LOAD commands if the variable has been previously defined.

1352 If N\$ has been defined either directly or in a program each of the following is an acceptable command.

LOADN\$, 8	/N\$	Normal LOADS
LOADN\$, 8, 1	?N\$	Special Memory LOADS
DOS 2.6	DOS 5.1 «WEDGE»	

*Up to 16, since that's the maximum number of characters in a file name or program name.

1353 `LOAD"N$,",8` will cause the system to try to load a program literally named N\$ and not whatever N\$ was defined to be.

1354 `LOADN$+"*",8` will cause the system to load the first program it finds whose name begins with whatever N\$ is defined to be, i.e. pattern matching in effect. (1340), see also a basic manual on "string operations."

1355 **SAMPLE PROGRAM:**

Rem: Sorry — you'll have to type this one.

Load and Run: `VARIABLE LOAD`

When the prompt appears, type the name of any PROGRAM on "Friendly Floppy".

```
10 PRINT"#####NAME OF PROGRAM TO BE LOADED####"
20 INPUTN$
25 PRINT"#####WILL LOAD "N$
30 LOADN$,8
```

Rem: I didn't include it because too many weird things happened. You will, of course, notice that it seems to automatically run the program. I've also gotten some syntax errors on lines that don't exist in the program it was supposed to load. Here's another "can of worms" you may wish to pursue. An avenue to explore: What all is "cleared" by a run command? The answer may lie in a study of machine language.

1360 **Summary Name Find**

1361 Check Documentation. Program user instructions should at the very least contain complete loading instructions including the name of the program.

1362 Check DIRECTORY for likely candidates.

```
LOAD"$",8                      or                      @ $
LIST
```

DOS 2.6
(see 1210 for details)

DOS 5.1 «WEDGE»
(see 1250 for details)

1363 Try Pattern Matching and Wild Card Techniques as described in 1340 and 1350.

(Continued)

1363 (Cont'd)

NOTE: `LOAD "*" , 8` will LOAD the 1st program on the disk (if none has yet been loaded).
Frequently, that will at least get you a menu or instructions.

REM: This is a quick reference section only. Complete "how to" descriptions are in sections
sited.

1370 Alternatives to Program Loss

REM: There isn't really one but, if the program in computer memory is that important, read
on.

1371 If your WEDGE is active and all you want to do is "view" the directory, `Type: @ $` or `> $`.
(All you'll get is a syntax error if your WEDGE is not active. You won't have hurt anything.

1372 If your program has already been SAVED to DISK or TAPE, don't worry about it. You can
always reLOAD it later.

1373 If you have a cassette and don't want to save to DISK, SAVE to cassette.

1374 If your program is too long to retype, then if you have a printer, LIST program on printer.

```
OPEN1,4
CMD1
LIST
```

```
PRINT#1
CLOSE1
```

1375 If your program is not too long to retype, type: `LIST`
and hand copy. (UGH!)

1376 If you are really daring, learn to Format a disk 2000 and then to SAVE your program 3000.
If you're a beginner, now is not really the time for that.

1377 If you want to learn to SAVE to disk before you learn to LOAD (not recommended) GO
TO 3000.

1378 If your program seems worth keeping, GO TO 1371.

1400 GENERAL FORMATS*

REM: This page is offered more to help you understand other commands than for any other reason.

DOS 2.6

LOAD"*dr:program*",*dv,sa*

parameters:

dr: drive number [0 or 1]
omit or use 0 with single drive system

program exact name of program - required
16 character maximum

dv device number [8-11] - required
8 for unaltered drive,
9,10,11 for others

sa secondary address [0 or 1]
omit or use 0 for normal LOAD
1 for special memory LOAD

<WEDGE> DOS 5.1

/program Normal LOAD
dv = 8, *sa* = 0

%program Special memory LOAD
dv = 8, *sa* = 1

↑program Normal LOAD and RUN
dv = 8, *sa* = 0

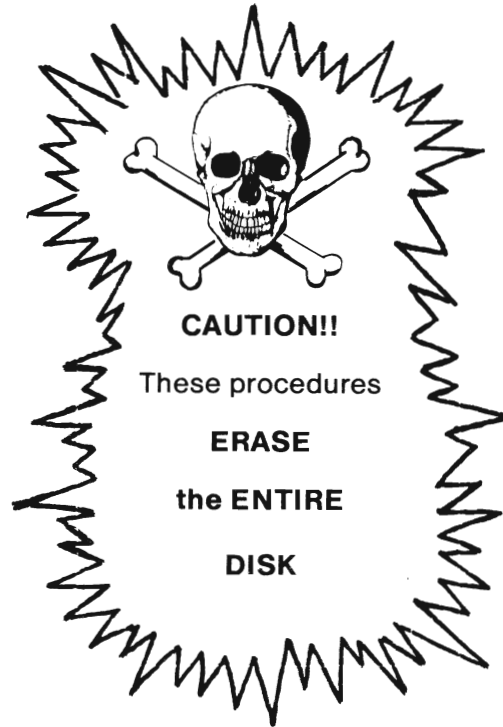
note- *dr:* drive number [0 or 1]
(may be specified as above)

* - Dual Drive — See appendix.

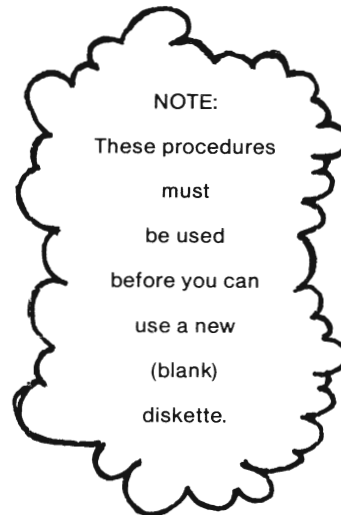
2000 FORMAT A "NEW" DISK
(or erase an old one)

CHAPTER DIRECTORY

2100	PRELIMINARIES
2110	What it means and what it does
2120	What you need and how to get it
2200	PROCEDURES
2210	How to "format a disk"
2220	Check it out
2230	Shorter ways
2300	ANALYSIS AND EXPERIMENTS
2310	Analysis of OPEN
2320	Analysis of PRINT#
2330	Experiments with OPEN
2340	Experiments with PRINT#
2400	GENERAL FORMATS



2010	PRE-FORMAT ("NEW") CHECK LIST	
		FOR HELP
2011	SYSTEM SET UP	: GOSUB 300
2012	SYSTEM ON	: GOSUB 400
2013	DRIVE READY	: GOSUB 512
2014	DISK SELECTED	: GOSUB 2121
2015	DISK INSERTED	: GOSUB 510
2016	NAME CHOSEN	: GOSUB 2122
2017	I.D. CHOSEN	: GOSUB 2123
2018	DOUBLE CHECK DISK	: GOSUB 2121



2100 PRELIMINARIES

Rem: You may wish to postpone this chapter. There is absolutely no reason why you can't learn some SAVE procedures (3000) before you try these. If you have "FRIENDLY FLOPPY" and a program, you have what you need to run through the SAVE chapter.

You will need to learn these procedures when you want to use brand NEW, BLANK disks.

2101 SUGGESTION:

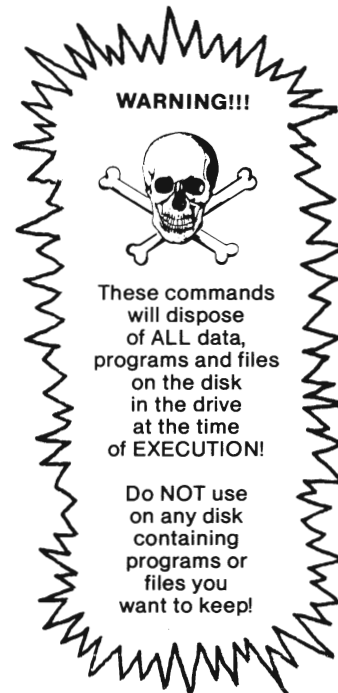
Learn to deal with the flashing red light before you get involved with sending commands to the drive. GOSUB 9000.

2102 SUGGESTION:

If you have a program in memory, I recommend you save it before attempting these procedures. You can "format" a disk without touching the program, but just in case... GOSUB 3000.

2103 SUGGESTION:

If you do not own a BLANK disk you have no real need of these procedures.² If you want to learn about the command, buy some BLANK disks. GOSUB *COMPUTER STORE*.

2110 **What it means and what it does**

Rem: You need not worry about mastering all of this information at this time. Anyone who can type commands on the keyboard can "format a disk."

2111 Before the drive can use a disk, it has to "set up housekeeping." Each disk drive system, be it our 1541 or another COMMODORE® drive, or a drive from a totally different computer, has its own format on "sector arrangement." See (233) for specifics on 1541 format.

2112 Suppose you have a friend who owns an entirely different system, say Brand X. You both go to the computer store and buy the same kind of diskette (single side, soft sectored, single or double density). So far these BLANK disks are identical. Each of you will have to "format" the disk. Your friend's system may put 10 sectors on each of the 40 tracks. Whereas your system puts differing numbers of sectors on 35 tracks. See (233) and the Appendix. This is the reason why you two can't trade disks.

² - Unless of course you already have a "beater" disk that you want to erase.

2113 Here are some of the things the DOS does when the commands to "format the disk" are executed.

- 1) It erases the entire disk.
- 2) It marks each of the 683 sectors (or blocks) with a starting mark and disk ID code in region 1 (235). Track and Sector numbers in region 2 (235) check points and timing gaps (to allow the drive to change modes between reading and writing operations) in region 1 and 4 (235).
- 3) Sets up the directory (which will include the name of the disk (260)).
- 4) Sets up the BAM (270).

Looks complicated but fortunately for us the system does all the hard work. All we have to do is type a couple of commands.

2120 **What you need and How to Get it**

Rem: To complete this chapter you'll need 3 things, a DISK to format, a Name for the disk, and an ID code.

2121 The DISK may be BLANK. (A brand new one, just home from the store), or an old one containing information that can be sacrificed (erased).

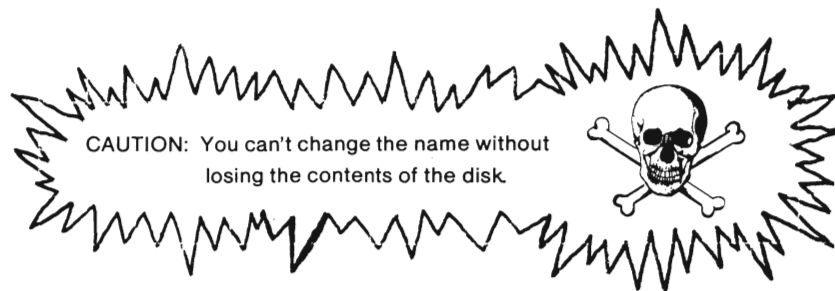
2122 The NAME

The name may contain up to 16 characters. If you try for more, the system will truncate (chop off) those that it cannot use.

Exactly 16 characters are reserved in the Directory header for the name of the disk. You will gain nothing by using shorter names.



The name you choose cannot be changed later without losing the contents of the disk. (If this is merely a test run, anything will do. The name is easy to change if you don't care about the contents).



(Continued)

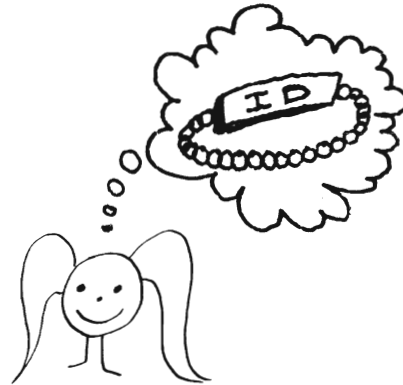
2122 (Cont'd)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
J	O	H	N	,	S		G	A	M	E	S					
J	,	S		G	,	S										does not save space
G	R	A	P	H	I	C	S		B	Y		J	A	N	E	
H	O	U	S	E	H	O	L	D		F	I	L	E	S		
D	I	S	K		1	*	◀									may use graphics characters
																may use all blanks
T	H	A	T		O	L	D		B	R	O	W	N	D	O	G ← will be truncated.

2123 ID CODE

Use exactly two characters. If you try for more the system will truncate. If you use only one, weird things happen.

The ID code you choose will be written on each of the 683 blocks on the disk. You will NOT be able to change the ID code without erasing the entire disk.

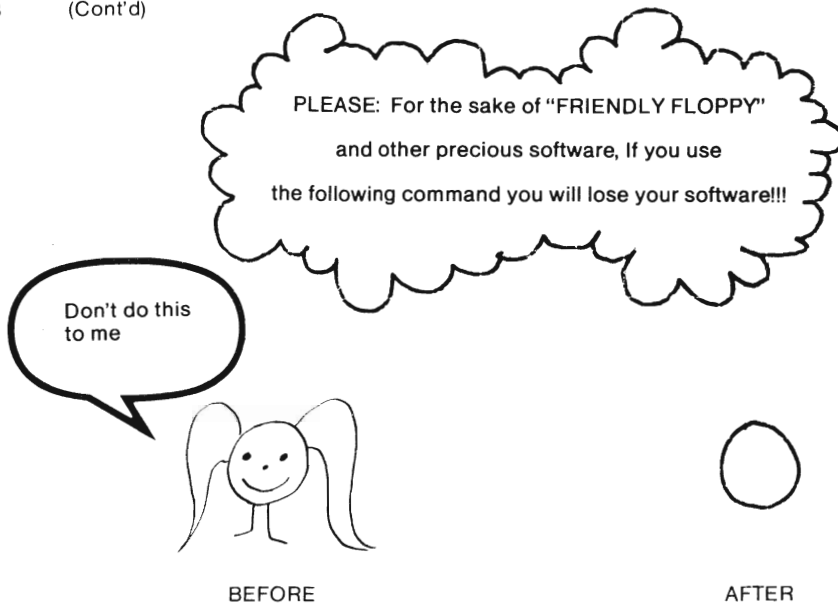


Each disk you use should have its own ID code. The DOS checks the ID code whenever it reads a sector. An error condition results when it fails to match ID codes. That may sound bad at first but can save your bacon. If you have to interchange disks frequently to get a job done, it's not too hard to get them mixed up. If two disks have the same ID code the drive can get mixed up. Precious files and/or programs could be lost.

To prevent foul-ups, give some serious thought to ID codes before you get into a mess. To simply learn the mechanical procedures, pick any old code. (See also 264 - Directory assistance).

(Continued)

2123 (Cont'd)



2200 PROCEDURES

Rem: This section is strictly a "how to." If you are not comfortable with that sort of thing, you may want to go through section 2300 first. Nothing in that section will get you into trouble if you pay attention. On the other hand, if you would rather "do it" first, then read about it, go on with this section.

2210 HOW to "format" a NEW DISKETTE

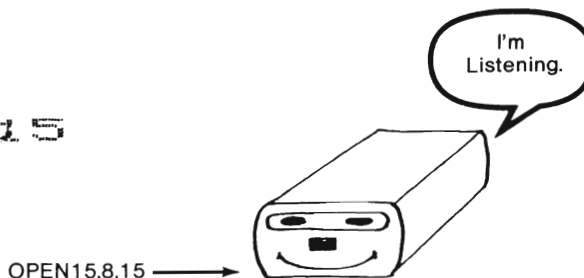
Rem: There are several ways to end up with a formatted disk. We'll go through one method in detail. It is not the easiest but it will give you more experience.



2211 Perform PRE FORMAT CHECK (2010).

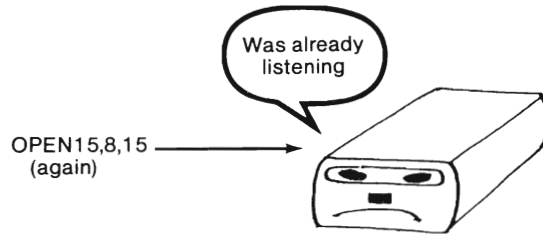
2212 Type: `OPEN15,8,15`

Looks like nothing has happened, doesn't it? The computer just says "ready" and flashes it's cursor.



(Continued)

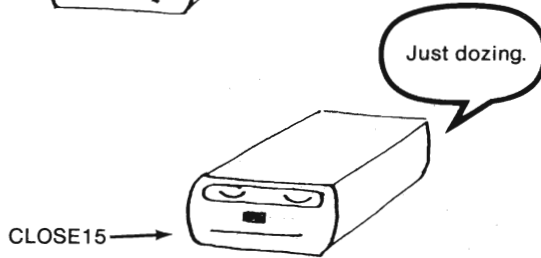
2212 (Cont'd)



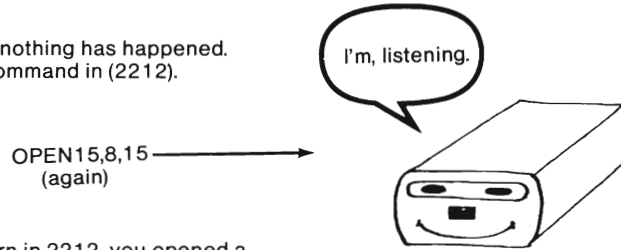
2213

If you feel like a little "adventure," retype the command in (2112). Notice the error message. (When you write your own programs, you must avoid opening the same file more than once).

Type: `CLOSE15`

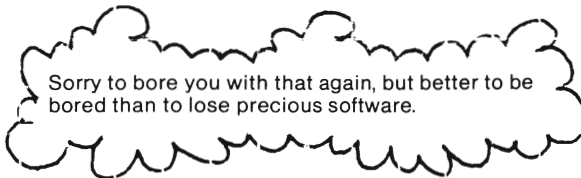
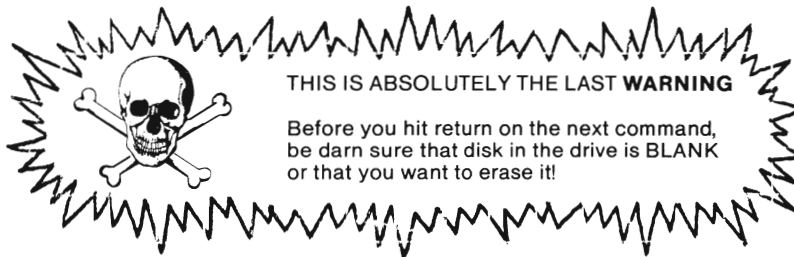


Again it looks like nothing has happened. Now, retype the command in (2212).



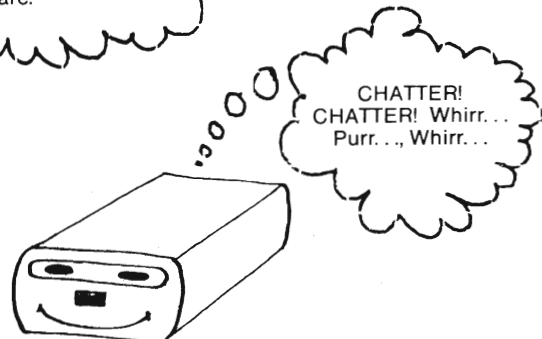
2214

When you hit return in 2212, you opened a communication link to the drive. It is now ready to receive commands and act upon them.



2215

When you hit the RETURN after the next command, all sorts of things will happen. The red light will come on and the drive will make NOISE. Be ready to wait. It has lots of stuff to write on the 683 sections it's going to form. (It only takes 80 seconds or so - time it.)

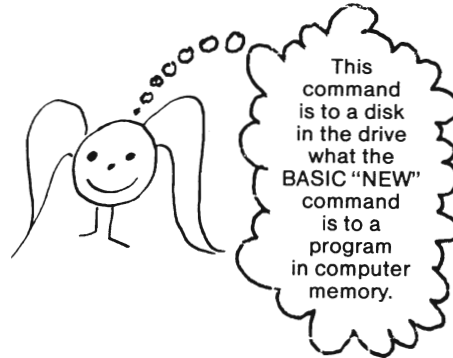


2216 Type: Your disk name Your I.D. code (2 characters)

```
PRINT#15, "NEW0 : _____, JJ"
```

2217 If all goes well, (and it should as long as there's a disk in the drive), when the red light goes out and the drive gets quiet, the disk has a format. It can now be used for your programs.

2218 Type: `CLOSE15`
 (To close communication link to the drive.)



2219 If the red light is flashing, type `␣`
 If you get a "syntax error" your wedge isn't active, go to 9000.

2220 Check it out

Rem: Let's see what we've accomplished.

2221 If your wedge is active, GOTO 2224.

2222 If you have a program in memory that you cannot sacrifice, read without doing

2223 Load the directory and LIST. GOTO 2225



2224 View the directory.



2225 You should see something like this.

```

      THE NAME      THE I.D. CODE
      ↓            ↓
  0  ██████████  ██████████
  664 BLOCKS FREE.
      ↑
  The DOS version & format type.(automatic)

```



2226 Time out for arithmetic.

683 blocks on the disk
 -664 blocks for your data and programs

19 blocks for directory and BAM.

See 0270 for more details.

2227 Rem: At this point you may wish to learn to SAVE, if you haven't already done so. The next section gives some shorter commands for formatting disks. They are definitely worth looking at. The section after that (2300) will give you a feeling for disk commands.

2230 **Shorter Ways**

Rem: I very cleverly used DISK NAME and ID (at the risk of insulting your intelligence. You are to substitute your choices.).



2231 A three step process is used in each of these.

- 1st: Open the command channel to the drive.
- 2nd: Send the "New" command to the drive.
- 3rd: Close the command channel.

```
OPEN15,8,15
PRINT#15,"NEW0:DISK NAME,ID
CLOSE15
```

or

```
OPEN15,8,15
PRINT#15,"N0:DISK NAME,ID
CLOSE15
```

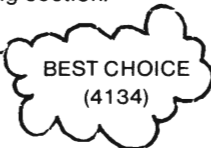
or

```
OPEN15,8,15
PRINT#15,"N:DISK NAME,ID
CLOSE15
```

Method of preceding section.

2 letters saved.
 N=NEW

The 0 can be omitted in a single drive system. (See 150)



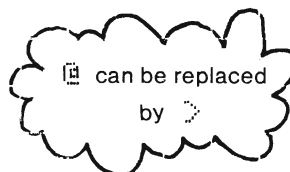
All of these can be used in program mode as well as in direct mode. If you do use in a program be sure to put a strong user warning in front of them, directing the user to change disks. Any disk that contains a program to "format" should have its write protect notch covered.

2232 These one-line commands will do the same thing as the first two steps of the commands above.

OPEN15,8,15,"NEWØ:DISK NAME,ID"
 or OPEN15,8,15,"NØ:DISK NAME,ID" Follow each with: CLOSE15
 or OPEN15,8,15,"N:DISK NAME,ID"

2233 DOS 5.1 «WEDGE» COMMANDS²

@N:DISK NAME,ID



2300 ANALYSIS AND EXPERIMENTS

Rem: The "analysis" sections pick apart the commands used to "format a disk." They are "read - not do" sections.

The "experiment sections are safe. In fact, you don't even need a disk in the drive to do them.

SUGGESTION: If there is a disk in the drive, take it out in case you are overcome with an uncontrollable urge to type.

2310

ANALYSIS of OPEN15,8,15

FILE NUMBER
 DEVICE NUMBER
 CHANNEL NUMBER

These numbers need not be the same. Many people find it simply easier to do so.



2311 FILE NUMBER - Strictly speaking any number from 1 to 255 can be used but stick to those less than or equal to 127. (128 through 255 are used for printers that don't have an automatic line feed after each return).

2312 DEVICE NUMBER - The number 8 is used to address an unaltered drive. Recall that 8 was used in LOAD commands. You could use OPEN15,9,15 if you wanted to open a link to a second drive. OPEN15,4 opens a communication link to the printer (160). More about this later.

2313 CHANNEL NUMBER - In general any number from 0 to 15 inclusive can be used in an open statement, but

15 is the *command (or error) channel* to the disk drive. ALWAYS use 15 to send commands to the drive.

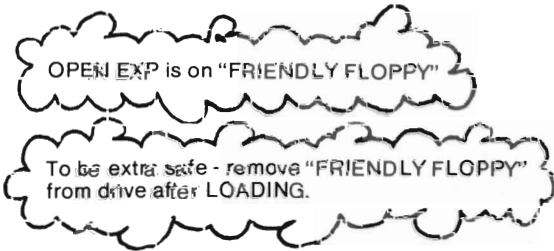
2 - 14 are for data files and 0 and 1 belong to the DOS. See (5310) for more details.

² - You can use NEW or NEWØ.

2330 **EXPERIMENTS with OPEN**

Rem: All you need for this section is your computer and monitor. If you really hate to type, power up your drive and have "FRIENDLY FLOPPY" handy.

2331 Type: (the following program) DOS 2.6 «wedge» DOS 5.1
 NEW if other program in memory. OR LOAD "OPEN EXP",8" or /OPEN EXP



5 REM***OPEN EXP² ← If you are typing skip the REM and use ? to PRINT

```

10 FOR I=1 TO 15
20 PRINT "WILL OPEN " I,
30 OPEN I,8,I
40 PRINT "OPEN"
50 NEXT
```

← This line will OPEN files as in lines earlier sections.
 Yup! You can use VARIABLES in open statements so long as they are defined. Line 10, of course, takes care of this for us.

Rem: Lines 10, 20 and 50 are just BASIC programming lines. If you don't understand them, you can do these experiments anyway, but you'll find the last few chapters very difficult until you improve your BASIC.

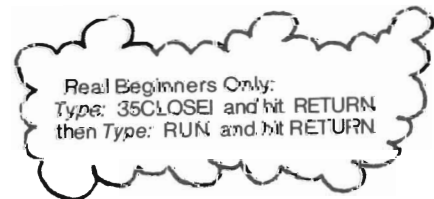
2332 Type: RUN and don't laugh when you see the error message. This isn't baseball. It was supposed to happen

How many "files" can you open. (Correct answer is 10).

2333 Add this line. Sorry you'll have to type now.

type: 35CLOSEI and run again.

See the Difference? All 15 are open with no error message. Hmmm!



file number >127
 ↕

² - PRINTER People. The double space listing was gotten with OPEN200,4:CMD200:LIST

2334 Change line 10 to read:

`10FOR I=0TO15` and Run. Now we have another error message. If you figured out why, give yourself a gold star. If not, re-read (2313) to the bitter end.

2335 Change line 10 to:

`10FOR I=0TO___` Play with this for awhile. What is the largest number you can put there before you get an error message? What's the smallest number that produces that error message?

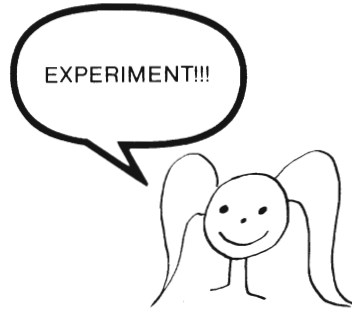
2326 Play with the numbers in line 30. Try this for a start:

`30OPEN1,8,I` with and without line 35.

`30OPEN1,I,I`

etc.

etc.



2337 Note: In practice, the type of files in use and the requirements of the peripheral devices that may be on, can limit the number of open files that will function.

2340 **EXPERIMENTS with PRINT#**

Rem: This section involves your computer and monitor only. Although I've used only direct mode commands you can use these in programs. (You probably won't want to, however). We are going to explore a really dumb way to put some words on the screen. Our objective is really to explore peripheral device command sequences.

2341 Type: `OPEN14,3`

If you get a "file open" error message, Type: `CLOSE14` and then repeat `OPEN14,3`

2342 Type: `PRINT#14,"FLOPPY"`

Wasn't that exciting? The word "FLOPPY" appeared on the screen.

You're right `OPEN14,3`
`PRINT#14,"FLOPPY" = PRINT"FLOPPY"`

type: `PRINT#14,"_____"`

Repeat as often as desired.

Compare to: `PRINT"_____"`



2343 Type: CLOSE14

Now try PRINT#14, " _____ "

Obviously the file is not open because we closed it in (2343).

2344 Explanation:

Device Number (3 for screen, (160).
 ↓
 OPEN14, 3 (the screen requires no secondary address, disk drive does. ex. OPEN15,8,15)
 ↑
 file number

²PRINT#14, _____
 ↑
 file number - linked to device and secondary address numbers in the OPEN statement.²

CLOSE14
 ↑
 file number - links to parameters in OPEN statement.

Study the general formats
 on the next page. Review
 as necessary.



² - many PRINT# statements can be included between an OPEN statement and its CLOSE.

2400 General Formats to "FORMAT"

DOS 2.6

```
OPEN fn, dv, 15
PRINT#fn, "Ndr:diskname,id"
CLOSE fn
```

or

```
OPEN fn, dv, 15, "Ndr:diskname,id"
CLOSE fn
```

parameters:

dr: drive number [0 or 1]
use 0 in single drive system

fn file number [1 - 127]

dv device number [8-11]
8 for unaltered drive,
9,10,11 for others

15 channel number (secondary address)
15 required to send commands to drive

N N = NEW command

diskname 16 character max

id use exactly 2 characters

<WEDGE> DOS 5.1

@N*dr*:*disk name*,*id*



3000 SAVE

CHAPTER DIRECTORY

3100	PRELIMINARIES
3110	What's going to happen
3120	What you need and how to get it
3125	Legal program name
3200	PROCEDURES
3210	SAVE - NORMAL
3220	SAVE - SPECIAL MEMORY
3230	SAVE - with REPLACE
3240	SAVE - «WEDGE USAGE»
3300	RELATED OPERATIONS
3310	VERIFY
3320	BACK UP
3330	Backing Up to Tape
3340	LISTING
3350	PROBLEMS with REPLACE
3400	GENERAL FORMATS

3010	PRE SAVE CHECK LIST	For Help:
3011	___ System Set Up	: GOSUB 0300
3012	___ System On	: GOSUB 0400
3013	___ Drive Ready	: GOSUB 0512
3014	___ Disk Selected	: GOSUB 3120
3015	___ Protect Notch Uncovered	: GOSUB 0221
3016	___ Program in Memory	: GOSUB 3124
3017	___ Disk Inserted	: GOSUB 0500
3018	___ Name Chosen	: GOSUB 3125

3100 PRELIMINARIES

3110 WHAT'S GOING TO HAPPEN

You're in system's physical motion to the SAVE command is similar to its reaction to LOAD.

When the SAVE command comes thru, the red light goes on, the drive makes noise and the computer says:

SAVING PROGRAM NAME

Meanwhile, the DOS and controller are very busy. The drive has to do all of the following:

- Make sure a disk is in it.
- Make sure the write protect notch is not covered.
- Check the directory (260) to see if there is room for another entry (only 144 allowed).
- Check the directory to make sure that the name chosen is not already used.
- Check BAM (270) to make sure there is room for another program block.
- Write the block and record location.
- Check BAM for another block, record location and write until finished.

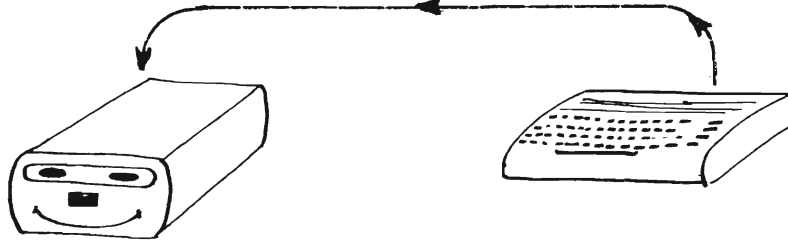
If any of the tasks above cannot be completed the drive will let you know by flashing its red light. (9000)

If all went well the red light goes out, the drive gets quiet and the computer, of course, says READY. If your WEDGE is active you'll even get a message from it.

(Continued)

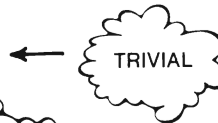
3110 (Cont'd)

It may seem like you could take a jog around the block while all of this is going on. Try it. If you can beat it, quit fooling with these computers and go to the Olympics.



3120 **What you need and how to get it**

3121 The system. If no system, go to dealer.



3122 A DISK that has been formatted.

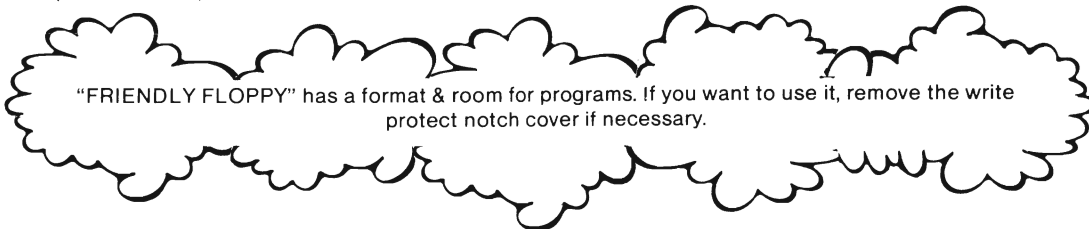


If you just got a box of BLANK disks, they have not been formatted (250). Before a BLANK disk can accept a SAVE command it has to be formatted. If you need to format a disk, GOSUB 2000.

If you are not sure of the development stage of your floppy, you can do either of the following.

(a) If your WEDGE is ACTIVE (1230), simply ask to view the directory. TYPE: `Q$` or `>$`. If your disk has a format, you'll see the directory. If not, you'll hear a "chatter" from the drive and the red light will flash. TYPE: `Q`

(b) If your WEDGE is NOT ACTIVE (`Q` results in syntax error), you can go ahead and try the SAVE. If your disk has no format, you'll have to deal with a flashing red light, but you won't hurt anything. (GOSUB 9000)



3123 ROOM on the disk for your program.

If you have a "baby floppy" that has just been formatted, you've got more room than you need. You need less than 160 blocks free to store a BASIC program that takes the entire memory of the C-64.

If your disk has been used for awhile, the simplest thing to do (if you don't want to check the directory yourself) is try the SAVE. If the disk is full the system will let you know. (Flashing red light(9000)).

3124 A PROGRAM TO SAVE

If you are simply trying to learn procedures or check out equipment, use an exciting program like the one below and SAVE it with an exciting name like "XXX."

10 PRINT"LEARN TO SAVE"



You could also simply load a program from tape or disk (1000 for disk load commands) and save that one.

3125 A LEGAL NAME

Last but not least, your program needs a name. For a successful SAVE to DISK, a name must be specified or a "file name omitted" error will result.²



The name must be *unique* (not already used on this particular disk).

The name can be up to 16 characters in *length*. You can type more but extras will be truncated, (chopped off). Theoretically you can use any characters you want. In practice, *avoid* the following:³

- ⌘ which is used for the directory. Instead of the usual error message, you may think you have a SAVE when you don't.
- ? or any number of ?'s. This is your wild card symbol (1340). Depending upon the location of your program you may never be able to load it again without scratching (4500) half the programs on the disk.
- * or any name ending in *. This is the pattern match symbol and can cause problems similar to those above.
- ⌘ ⌘ : ⌘ : ⌘ : are all WEDGE command signals and can cause trouble when your WEDGE is active.

² - You can omit the NAME on tape but NOT on disk.

³ - This may not be an exhaustive list.

3200 PROCEDURES

Rem: The SAVE procedures are so similar to those for LOAD so the explanations are not nearly so detailed. If you need more information refer to the LOAD sections noted.

3210 SAVE - normal memory²

3211 Perform Pre SAVE check

3212 Type: `SAVE "XXXX", 8`

Name of program.
Substitute your choice.

Device Number
If omitted, the system
will try to save to tape.
Hit RUN/STOP and start over.

XXXX stands for the name of the program.
"PROGRAM NAME"
was not used since there is already
one on "FRIENDLY FLOPPY" by that name.

3213 If the red light is flashing, GOSUB 9000.

3214 If all is well, note program name exactly for future reference.

3220 SAVE - Special memory³

Rem: In order to load a program to special memory it must have been saved in the following manner. (If you are wondering if your program needs this, it probably doesn't! If you have to ask, you can't afford it).

3221 Perform Pre SAVE Check

Your
choice

3222 Type: `SAVE "XXXX", 8, 1`

If you have programs that require this
it would be wise to code the name in
some manner.

3223 If red light flashing, GOSUB 9000

3224 If all is well, note name for future reference.

² - If you are saving an unfamiliar program and it won't run properly after a normal save, try (3220)
³ - Programs saved in this manner may require special memory load (1260).

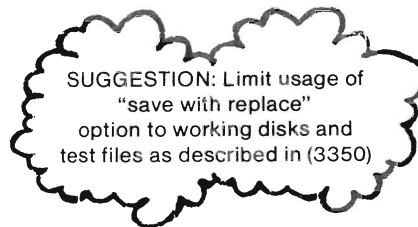
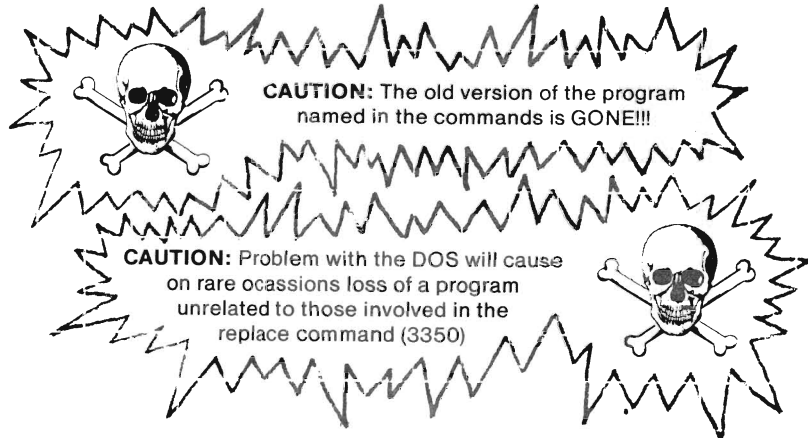
3230 SAVE with REPLACE

Rem: To the die-hard programmer, revisions are a way of life. Disks, no matter what their capacity, would soon fill up if every version needed its own blocks and directory entry. DOS comes to our rescue with two options:

- 1 SCRATCH (4500) then SAVE — Safest
- 2 SAVE with replace — Dangerous

Of the two, the SAVE with replace is the most convenient, but on rare occasions will lose a previous program in the process². (3350)

Essentially the "save with replace" does exactly that—replace an old version of a program with a new one. The new program need not even be related to the old one. After execution the directory looks about the same as it did before. The new version's name may appear in a different place and any change in length will be reflected. The directory notes the new track and sector starting point. BAM marks the old blocks "free" and the new ones "used." (260-270)



Rem: This option will be described in detail in spite of the DOS problems with it. You may wish to read (3330) before you test these commands.

² - this is a DOS problem. See *COMPUTE!'s GAZETTE*, October 1983, "DOS WOES," Page 14.

3231 Perform Pre Save Check up to 3018. This time you need the exact name of the program to be replaced. (The disk you select must of course contain that program.). If you are ready, GOTO 3233.

3232 If you are not sure of the name of the program or whether or not the disk in the drive contains that program, read on.

If your WEDGE is ACTIVE (1233),  will show you your directory.

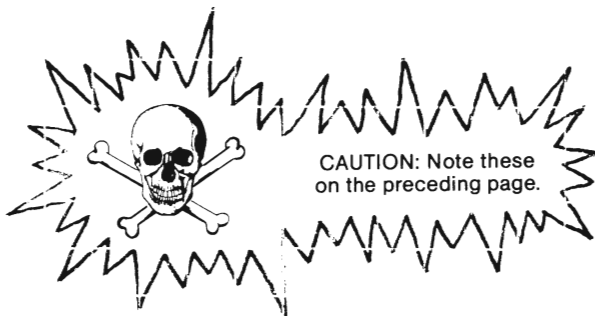
If your wedge is not active here are some options:

(a) SAVE the version currently in memory under a bizzare name. LOAD and LIST directory and decide what to do. When you want to get rid of the extra version, see SCRATCH.

(b) SAVE under the name you think is there. If the entry is on the disk you'll have to deal with the flashing red light (9000). If it wasn't on the disk your program is now saved.

(c) Go ahead with the save with replace option. This time the error condition will result if the program is not on the disk. You can then save normally. If the program was on the disk, its been replaced by the one in memory.

Rem: If these things are happening to you, it's time to get organized (see 262)



3233 Here is a list of the commands. XXXX is the name of the program on the disk to be replaced by the one in computer memory. The "@" is the signal. The colons are required.

Normal memory "save with replace" commands.²

SAVE"@:XXXX",8 The 8's are the device number that signals "disk drive."

SAVE"@0:XXXX",8 This 0 is the DRIVE number. BEST CHOICE in any case. (3360)

SAVE"@:XXXX",8,0 These 0 are the secondary address that signal "normal memory" and are optional.

SAVE"@0:XXXX",8,0

↑
— This is the drive number and is optional in a single drive system (3360)

(Continued)

² All equivalent (3402)

3233 (Cont'd)
Special memory "save with replace" commands.³

SAVE"@:XXXX",8,1 These 1's are the secondary address that signal "special memory"
and are REQUIRED or a normal memory SAVE will take place.

SAVE"@0:XXXX",8,1

↑
└ Drive number (see preceding page).

3234 After you type your chosen command, the system will respond as with the usual SAVE command. The "@" or "@0" will not appear in the directory.

3235 If the red light is flashing, GOTO 9000

Suggestion: Use with caution: (3350)

3240 **SAVE USING THE WEDGE** DOS 5.1 Supporting DOS 2.6

Rem: Allow me to dispense with "step-by-step" bit. To the best of my knowledge the wedge does not include a command for a special memory save.

WHEW!

NOTE: " ← " is the left arrow key at the upper left hand corner of your keyboard—not cursor left.

3241 ←XXXX is equivalent to SAVE"XXXX",8
or SAVE"XXXX",8,0

NORMAL
SAVE

3242 ←0:XXXX invokes the replace option and is equivalent to normal saves in (3233)

SAVE
WITH
REPLACE

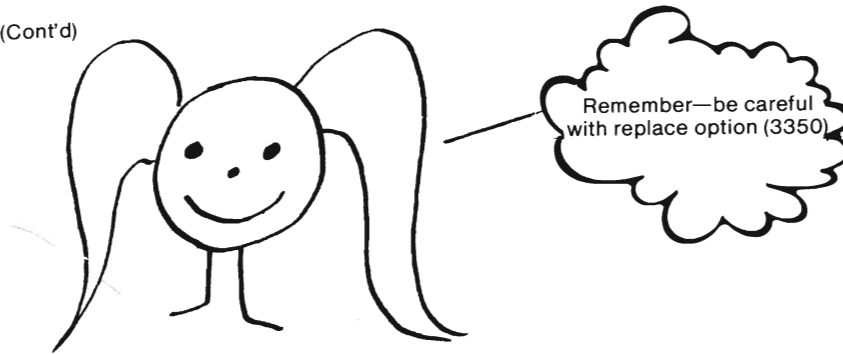
Rem: DOS 5.1 will initiate the save operation and even read the error channel for you. When you see

00, OK,00,00

the job is done.

³ Both equivalent (3402)

3242 (Cont'd)



3300 RELATED OPERATIONS

Rem: FLOPPIES are delicate. Even if you take care of them "by the book" things can go wrong. Here are some of the things you might want to do after you save a program

3310 VERIFY

Rem: The VERIFY command causes the system to compare a program stored on a disk in the drive with the program in computer memory. The "OK" appears only when the two programs match BYTE by BYTE*.

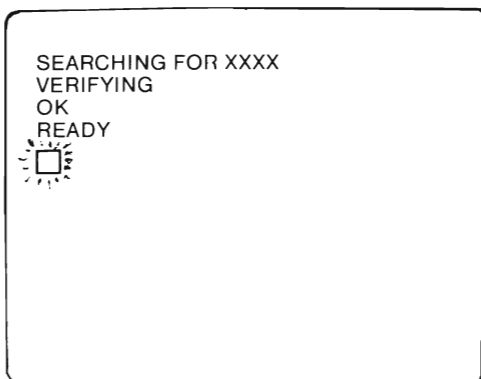
Rem: The probability of an error after a save to disk is much much lower than after a save to tape. You may simply want to use VERIFY to see if you've saved a revision.

Here's the command. XXXX is again the exact name of the program you wish to verify.

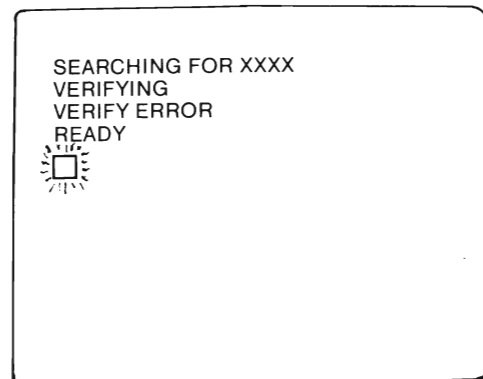
3311 Type: `VERIFY"XXXX",8`



3312 After you send the command the drive goes to work. (Solid red light and noise). As the process is completed, the computer says:

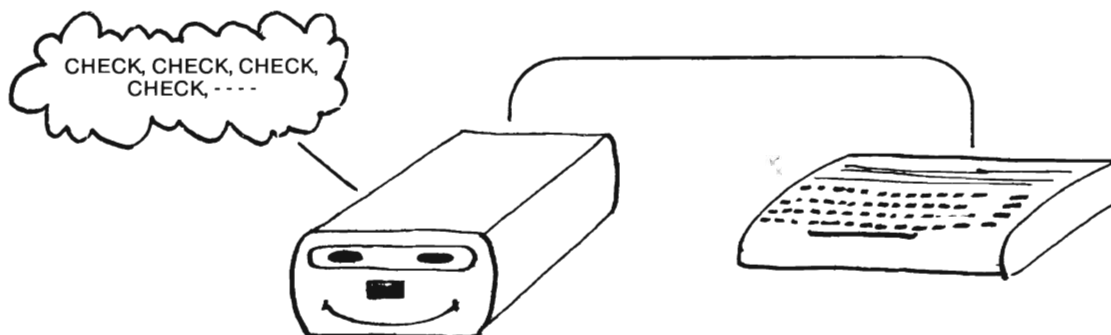


or



* LIMITATION: Programs saved from an expanded VIC-20 may not verify when LOADED to a different expansion. Not to worry, they'll run so long as the program runs on both.

3313 If the red light is flashing GOTO 9000



3320 **BACK UP COPIES**

Rem: It is wise to make BACK UP² copies of important software. A power outage or blown fuse can damage a disk. Even a single messed up BYTE can ruin a program³.

Most Copyright covered software at least allows the user to make a back up copy for personal use. Machine language programs require methods beyond the scope of this text².

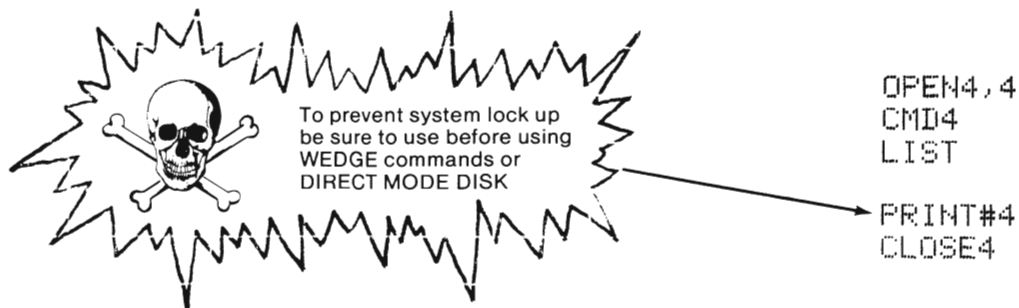
The easiest way for the normal user is to simply SAVE on two different disks.

3330 **BACK UP on TAPE**

It wouldn't hurt to back up really important programs on tape if you have the equipment. Programs that don't involve disk files and operations can be run and worked on even if your drive goes "down."

3340 **LISTING**

If possible keep a LISTing of important programs. If you have a lot of long programs, a PRINTER may be your next investment. (If you already have a printer, own stock in a paper factory and use the following commands in direct mode.)



² "For want of a nail... the kingdom was lost."

³ - Programs that will copy an entire disk are available. Use DUPLICATE if you have a dual drive system. (0150).

3350 The PROBLEM with the REPLACE OPTION.

```

DIRECTORRY
XX Program A
YY Program B
    
```

When we ask to load A, we get B. It seems that DOS thinks A is located where B is located. The problem occurred during a save with replace of B. B operates normally.

After using the replace option hundreds of times with no problem, I've found that I've lost at least two programs. Each appears in the directory with the proper length indicated. Unfortunately all attempts to LOAD either of these result in LOADING the program that follows (as shown in the directory). Other 1541 owners report the same problem².

Although the "replace option" seems to work beautifully 99% of the time, I have sadly dispensed with its services. During program development stages, I use a "working disk" and use a regular SAVE version number². When the program reaches the "using" stage I SAVE it on a "Final Version" Disk. As the "Final Version" is revised, I SCRATCH (4500) the old final version and SAVE the new final version. From time to time I clean up the "working disk" with SCRATCH, too.

e.g.

```

WORKING DISK
GB 1
GB 2
GB 3
GB 4
GB 5
GB 6
GB 7
GB 8
    
```

```

FINAL VERSION DISK
GRADE BOOK
--
--
--
--
--
    
```

3351 The Bottom Line:

BACK UP ... BACK UP ... BACK UP ... BACK UP ... BACK UP ... BACK UP ...

3360 **AFTERTHOUGHT**

Rem: Having just suffered the loss of another program² I figured I'd better add an afterthought. To be honest the rest of this chapter is with the Typesetter and fixing the chapter from scratch will delay this text another month.

I cannot recall for sure, but I have the uncomfortable feeling that this loss was not the result of the replace option (3350). It may have occurred after a SCRATCH that left room for a program followed by a SAVE that would fit. I checked all disks that I own for duplicate ID codes (2123) and confess that I possess such a pair. Neither were involved in the process.

Here are the procedures³ I'm using when I have to be safe.

1st If disk has been changed, INITIALIZE (4300)

2nd Use longer command forms shown below.

SAVE"O:program",8 or **← O:program**

3rd Make back up copy immediately. (I scatter my back up copies over a multitude of disks. It makes a program harder to find but decreases the probability of the same program being lost twice).

I'm even thinking of saving the same program twice on the same disk unless they are simply development versions. (I don't really care if version 8 replaces version 7. It was going to anyway). This method will make for longer directories but seems to save some hassles.

² - Fortunately I had a back up. I try to practice what I preach.

³ - Somewhere in my faithful periodical* reading somebody mentioned drive number (0150).

* - Primarily "COMPUTE!" which is currently running a series on files. Too late for me but thank goodness they are confirming items you'll encounter later. And, "COMPUTE!'s GAZETTE" which has come a long way in six months. I recommend both of these without reservations.

3400 GENERAL FORMATS

DOS 2.6

SAVE"*dr:program*",*dv*,*sa*

SAVE"@*dr:program*",*dv*,*sa*

SAVE with replace - NOT recommended

parameters:

dr: drive number [0 or 1]
use 0 in single drive system

program exact name of program - required
16 character maximum
avoid wild card, pattern match,
and <wedge> symbols

dv device number [8-11] - required
8 for unaltered drive,
9,10,11 for others

sa secondary address [0 or 1]
omit or use 0 for normal SAVE
1 for special memory SAVE

<WEDGE> DOS 5.1

<*dr:program*> Normal SAVE

<@*dr:program*> SAVE with replace - NOT recommended

dr: as above
dv = 8, *sa* = 0



4000 OTHER COMMANDS

Rem: Descriptions of other commands that can be sent to the drive are given in this chapter. Although primarily a reference, this chapter deserves a 1st reading before you continue to files. Be sure to cover the first section since all other sections follow the patterns set. After your 1st reading take a look at (0070), (5000), (6200) and (6300).

CHAPTER DIRECTORY

4100	GENERALITIES
4200	NEW (New Use)
4300	INITIALIZE
4400	VALIDATE

} involves the
entire disk

4500	SCRATCH
4600	RENAME
4700	COPY (Simple)
4800	COPY (To append)

} for specified
data files and
programs

4010

Section Directory

4 X 10	Purpose
4 X 20	Command Sequences (DOS 2.6 and DOS 5.1 Wedge)
4 X 30	Notes

4100 GENERALITIES

Rem: (4100) refers to all sections in this chapter.

4110 Purpose: In general, a command to the drive causes it to do something (format a disk, rename a program, etc). The purpose of the command is explained in this location throughout the chapter.

4120 Command Sequences: The command sending procedures are the same for all commands. The abbreviations shown below will be used throughout the chapter. DOS 2.6 provides you with two options that can be used in both direct and program modes. If your «wedge» is active, you have a 3rd option in direct mode. I wish the «wedge» worked in program mode but it doesn't.

Abbreviations: fn = file number. (5312)
 (any number from 1 to 127 may be used).
 dv = device number (0160)
 (use 8 for unaltered drive)
 (9 is the usual choice for the second drive)
 15 = channel number (5312)
 (15 is ALWAYS used to send "commands" to the drive).
 dr = drive number (0150)

Command string: changes from command to command. Details in individual sections.

4121 DOS 2.6 (option 1)

Use option 1 when you wish to send several commands.

```

OPEN fn,dv,15
PRINT#fn,"command string"

PRINT#fn,"command string"
CLOSE fn
```

4122 DOS 2.6 (option 2)

```

OPEN fn,dv,15,"command string"
CLOSE fn
```

Use option 2 when your «wedge» is not active and you wish to send one command in direct mode. Or combine with option 1 in program mode as is shown in COMMAND DEMO (6300).

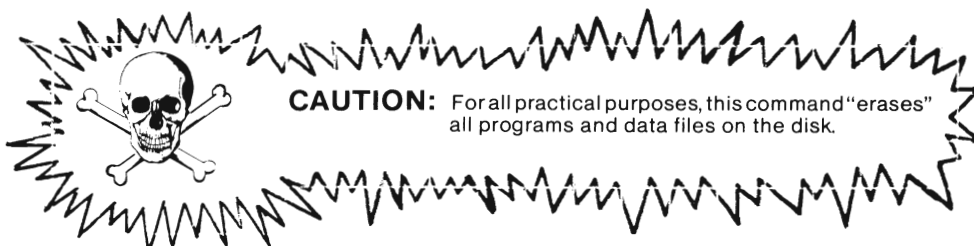
4123 «WEDGE» DOS 5.1 supporting DOS 2.6

```

@ command string
```

@ does all the work

- 4130 **Notes:** Further explanations and limitations that apply to individual commands are in lines 4X30. The following apply to all of the commands in the chapter.
- 4131 All commands in this chapter will cause an error condition (flashing red light) if there is no disk in the drive. (9302) provides an alternative.
- 4132 All punctuation is required except the last quote mark. That may be eliminated when it is the last symbol in a line as it can be in BASIC PRINT statements.
- 4133 All commands can be used "as is." You will, of course, want to substitute your own names.
- 4134 The use of the drive number, the O's, is 'fuzzy' to say the least. Most commands seem to work fine without it. As I continue to work with the system, I'm leaning toward using it when I need to be safe. Better to type an extra O than to mess up something important (3360). If you are using a 'dual drive' you must pay attention to the drive number.
- 4135 As you practice these commands, remember that you have to CLOSE before you can reOPEN. In direct mode all you get is an error message (no big deal). Use the PRINT# version since the file is already open or CLOSE fn. Be careful. Closing the command channel can make a mess. (See notes below).
- 4136 Programmers note: If you are debugging a program that involves data files and the program stops (error or run/stop) follow the procedures in (6253). Close data files before you close the command channel.
- 4137 Programmers note: In the programs in the following chapters, notice that the command channel is OPENed at the beginning. Follow the pattern "first open, last closed." Your system will get terribly confused if you CLOSE 15² before the data channels. The drive will think they are closed, your program will think they are open. I think this may be why I'm getting 70-no channel after I use wedge commands during an interrupt. This idea requires a bit more study and experimentation before I can say for sure.
- 4138 "COMMAND DEMO" (6300) shows most of these commands in action. You can safely use it before completing (5000), (6100) and (6200).
- 4139 In each case, the entire word can be used in place of the single letter shown.
- 4200 **NEW** (new use)
- 4210 **Purpose:** The NEW command used to completely format a BLANK disk (2000) has a short form that can be used to simply erase the directory and make BAM think that all 664 blocks are free. The process is much quicker. DOS does not change all the ID codes and a lot of the rest of the stuff it writes when a BLANK disk is formatted. This form of the command turns an old, mature floppy back into a 'baby' (0250). The ID code is retained. If you want to change ID codes you have to use the methods of (2000).



² - The file number of the command channel.

4220 **Command Sequences:** (N is short for NEW)

4221 DOS 2.6 (option 1)

```

OPEN15,8,15
PRINT#15,"N0:DISK NAME"
CLOSE15
    
```

your choice
↓

┌──────────┐
command string

the first number in each line is the file number (4220).

4222 DOS 2.6 (option 2)

```

OPEN15,8,15,"N0:DISK NAME"
CLOSE15
    
```

4223 «WEDGE» @N:DISK NAME

4230 **Notes:** (not used in COMMAND DEMO)

4231 Notice that the ID code symbols were not specified as they were in (2400). This is how the system knows it doesn't have to do all 80 seconds worth of work. (2110)

4232 You can either repeat your old disk name or choose a new one.

4233 The "drive numbers," the ϕ 's, can be omitted in a single drive system with this command. As I work with this system I'm leaning toward using them all the time or at least when I need to be safe.

4300 **INITIALIZE**

4310 **Purpose:** This command "loads" the directory and BAM into drive memory and must reset some parameters. It can be used to shut off both a flashing and solid red light (assuming there's a disk in the drive). (4331).

4320 **Command sequences:** (I is short for INITIALIZE)

DOS 2.6 (option 1)	DOS 2.6 (option 2)	«wedge»
OPEN15,8,15	OPEN15,8,15,"I"	@I
PRINT#15,"I"	CLOSE15	
CLOSE15		

4330 **Notes:** (program sample in (6341)).

4331 It is critical to use this command after changing disks especially if there is a chance that they might have the same ID code. Your system will get terribly confused if it tries to use the directory and BAM information about one disk on a different one. If it sees different ID codes it knows a change has been made.

4332 The solid red light and procedures to deal with it are covered in (6253).

4333 The command strings above have not used the drive number. Use "IØ" if you prefer (4334).

4334 Specifically, how this command goes about shutting off red lights I have no idea. It must reset some DOS parameters. That might be an interesting topic to pursue if you care. If you'll be satisfied with an over simplification, "it also makes the drive think it just woke up." That idea served me very well for quite some time.

4335 Programmers note. Although not used extensively here, it would be a good idea to build this command into your programs whenever the user is allowed to change disks.

4336 It does not appear to close data files when a write operation is interrupted. (5352) (6253)

4400 **VALIDATE**

Rem: I've not tested this command as thoroughly as some of the others. Some ideas for your own experimentation are presented.

4410 **Purpose:** This is a house keeping command. In contrast to the NEW command (4200), VALIDATE spares your programs, properly closed data files, your disk name, etc. It disposes of unclosed files and frees up odd blocks that may be left unused when a shorter program replaces a longer.

4420 **Command Sequence** (V is short for VALIDATE)

DOS 2.6 (option 1)	DOS 2.6 (option 2)	«wedge» ?(4433)
OPEN15,8,15	OPEN15,8,15,"V"	@V
PRINT#15,"V"	CLOSE15	
CLOSE15		



CAUTION: DO NOT use with this command if you think the disk may contain random files. (Files prepared with software of unknown structure) (4431)

4430 **Notes:**

4431 Your user's manual cautions you on the use of VALIDATE on disks that contain RANDOM files. I've used it frequently on the disk that contains RND SAMPLE. My records read back in the normal manner. I think they may mean that BLOCKS ALLOCATED but not used will be freed. To test this hypothesis you might try ALLOCATING some blocks, viewing BAM², sending this command and viewing BAM again. If you see a change, my hypothesis was correct. If not, it's back to the "drawing board."

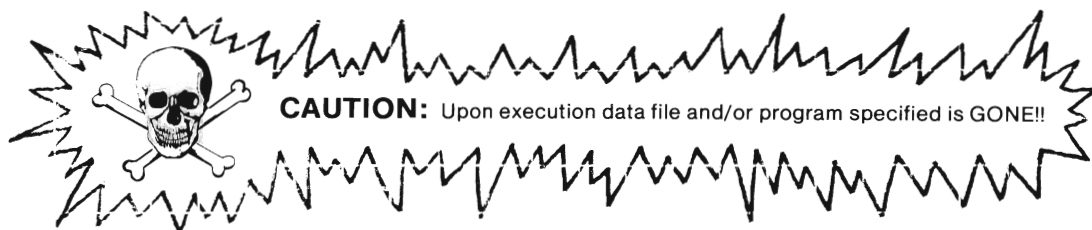
4432 The VIEW BAM test might be fun with a working disk. Note initial blocks used, send the VALIDATE, check again. Experiment!

4433 I have no documentation supporting the «wedge» command shown above. My drive makes the same noises for about 27 seconds as it does when I send the DOS 2.6 command. The same visible result (4334) shows up.

4434 One visible result of the VALIDATE command is that the open file marks in the directory disappear. Try it with (6253). You may of course specify the drive number by using "Vø"(4134)

4500 **SCRATCH**

4510 **Purpose:** This command will SCRATCH (essentially erase) data files and programs.



4520 **Command Sequences** (S is short for SCRATCH)

4521 DOS 2.6 (option 1) OPEN15,8,15
 PRINT#15,"S0:FILE NAME"
 CLOSE15

Use of drive number, 0's, is optional but recommended.

4522 DOS 2.6 (option 2) OPEN15,8,15,"S0:FILE NAME"
 CLOSE15

4523 «WEDGE» @S:FILE NAME

4530 **Notes:** (program sample in (6346)).

² - "VIEW BAM" is a program on your Test Demo Disk.

- 4531 If you read the error channel (9000) after a SCRATCH operation, the DOS will tell you how many files were scratched. This is not an "error" condition, simply information. COMMAND DEMO (6300) makes use of the message.

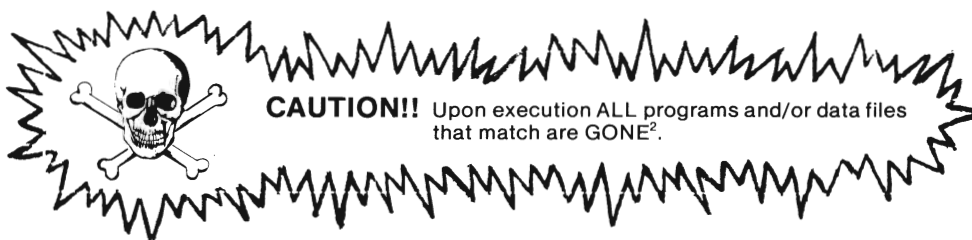
```
00, FILES SCRATCHED, XX, 00
```

↑
number of files scratched where the track number is usually located (9251).

- 4532 Pattern matching and wild cards can be used to SCRATCH more than one file and/or program at a time. (1330) and (1340) methods apply.

ex S:TE# will SCRATCH ALL files and programs with names that start with TE. TEMP, TELEPHONE, TEST, etc will be SCRATCHED.

S:F??D will scratch all files and programs whose names match. FIND, FEED, F2XD, etc. will be SCRATCHED.



4600 RENAME

- 4610 **Purpose:** (Guess what?!) This command will simply RENAME a data file or program. It is especially useful with data files when some sort of holding file is necessary.

4620 Command Sequences:

4621 DOS 2.6 (option 1) OPEN15,8,15
PRINT#15,"R0:NEW NAME=OLDNAME"
CLOSE15

4622 DOS 2.6 (option 2) OPEN15,8,15,"R0:NEW NAME=OLDNAME"
CLOSE15

4623 «WEDGE» @R:NEW NAME=OLD NAME

² - They are probably still on the disk but the elves can no longer get at them. (0270)

- 4630 **Notes:** (program sample in (6345)).
- 4631 Future reference to the program or file must be made to its new name.
- 4632 "Error condition" occurs if NEW NAME is already on the disk or if OLD NAME is not. It works like 62-File not found in Load commands or 63-File exists in SAVE commands.
- 4633 My favorite trick is to forget which name comes first. You may see that it works like a BASIC statement LET X = A, (or X = A for short). X is given A's value. "A" keeps its value. In the command above OLD NAME, as a name, no longer exists. The file is still there, it just has a new name.
- 4700 **COPY** (simple)
- Rem: The next section deals with another use.
- 4710 **Purpose:** (Guess what?!). This command will COPY a data file (or program) on the disk under a new name. Unfortunately, in a single drive system (0150), the original and the copy are on the same disk. This command could be used for creating a holding file. It's also of use when you want to change the location of a program or file in the disk directory.
- 4720 **Command Sequences:** (C is short for COPY)
- 4721 DOS 2.6 (option 1) `OPEN15,8,15
PRINT#15,"C0:NEW FILE NAME=0:OLD FILE NAME
CLOSE15`
- 4722 DOS 2.6 (option 2) `OPEN15,8,15,"C0:NEW FILE NAME=0:OLD FILE NAME
CLOSE15`
- 4723 «WEDGE» `@C:NEW FILE=OLD FILE`
- 4730 **Note:** (not used in COMMAND DEMO)
- 4731 If you need to make a copy of a program on a different disk, a LOAD, change disks, INITIALIZE (4300), and SAVE is the easiest.
- 4732 A Back up Program is your best bet if you want to copy an entire disk.
- 4733 I confess I haven't tried this one with a relative file and I doubt that it will handle random files.
- 4734 I have not tried the "COPY ALL" program on the Test demo disk because I simply don't have two drives. I "hear tell" that it works. Check it out if you have two functioning drives.

4735 The command will, of course, cause an error condition similar to that described in 4632.

4800 **COPY** (to append)

4810 **Purpose:** This command is used to combine sequential data files. Essentially it sticks one file onto the tail of another. This form of COPY can be used if all you want to do is add to an existing file.

4820 **Command Sequences:** (C is short for COPY)

4821 DOS 2.6 (option 1)

```
OPEN15,8,15
PRINT#15,"C0:NEW FILE=1ST OLD,0:2ND OLD,0:3RD OLD,0:4TH OLD"
CLOSE15
```

4822 DOS 2.6 (option 2)

```
OPEN15,8,15,"C0:NEW FILE=1ST OLD,0:2ND OLD,0:3RD OLD,0:4TH OLD"
CLOSE15
```

4823 «WEDGE»

```
@C0:NEW FILE=1ST OLD,0:2ND OLD,0:3RD OLD,0:4TH OLD"
```

4830 **Notes:** (program sample in (6344).

4831 Itsure would be nice if this relatively simple command could be used to append programs. Try it. All you get is the 1st old program.²

4832 Guess what?! An error condition results when. ..(4632)

4833 Four files are supposedly the maximum that can be combined in any one pass. Several passes will be necessary if more than four are to be combined. The length of the file names seem to dictate how many we can do. It might be interesting to do with string variables for file names. Appropriate +'s and quotes would be required. (6232) should give you the idea.

4834 COPY (to append) seems to want the DRIVE numbers (4134).

² - I need to know more about how this system especially the C-64 works. "Machine language" of which I confess nearly total ignorance is my next port of call.

5000 DATA FILES IN GENERAL

CHAPTER DIRECTORY

5100 TYPES OF FILES

- 5110 Sequential
- 5120 Random
- 5130 Relative

5200 COMMON PROCEDURES (overview)

5300 COMMANDS - GENERAL FORMAT & FUNCTION

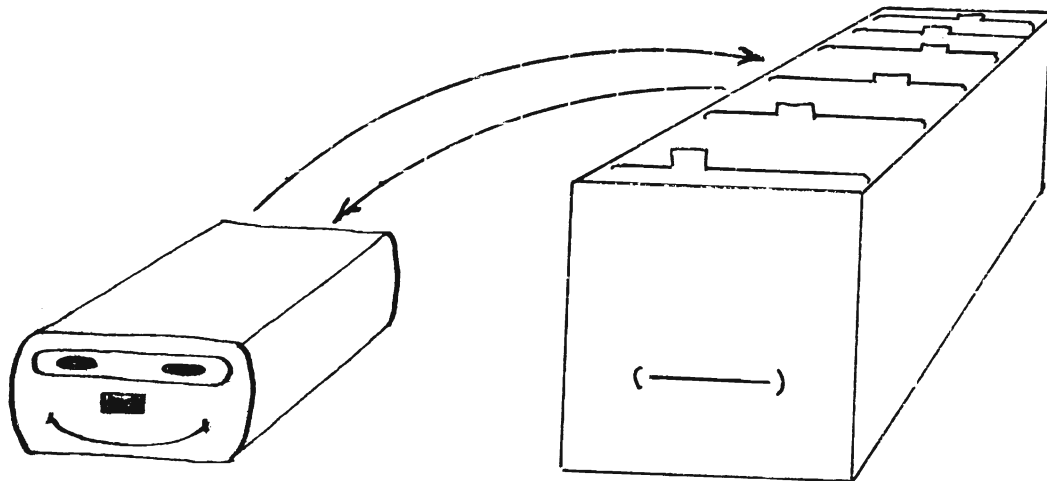
- 5310 OPEN
- 5320 CLOSE
- 5330 PRINT#
- 5340 INPUT# & GET#

5400 FILE DATA FORMAT

- 5410 The Problem
- 5420 Multiple PRINT#
- 5430 Separators
- 5440 PRINT# Punctuation
- 5450 Numeric Variables

Nature, Length, Advantages & Disadvantages

FILES



5100 TYPES OF FILES

REM: Your system can handle three types of data files: sequential, random, and relative. It also thinks of programs as files. In these chapters (5000 through 7000) we'll use the word "file" to mean "data file".

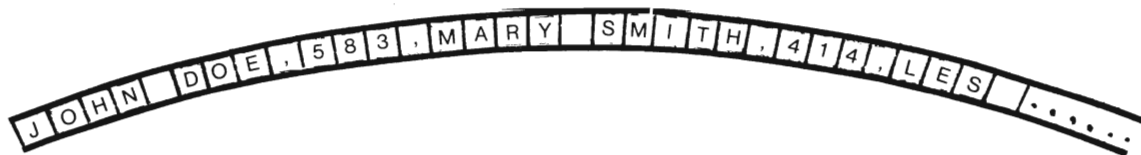
5110 Sequential Files

5111 The **nature** of a sequential file is exactly what the name suggests. A sequential file is written byte-by-byte from beginning to end. It can only be read in the same way — from the beginning. i.e. If you are interested in the 83rd entry, the system must read the first 82.

5112 The **length** of a sequential file is limited only by the capacity of the disk. Practical considerations intervene. As fast as your drive is, it still takes time to read an entire diskette.

5113 The primary **advantage** of sequential files is ease of use. If the task does not require a lot of skipping around, they are ideal. Programs that involve long series of data statements are prime candidates for adapting to sequential files.

5114 The **disadvantages** are due to their nature. Editing is the primary problem. Adding to the end of a file is fairly simple (4800). To change a single entry within the file requires rewriting the entire file. Unless the user wants to enter all the data, it is easiest (if memory permits) is to read into an array, make the changes, and re-write the file.



5115 A detailed discussion with specific command formats and sample programs can be found in Chapter 6000.

5120 Random Files

5121 The **nature** of a random file is also what the name suggests. With RANDOM FILES we can read or write to any block on the disk. There is even a way of accessing a single byte in that block.

5122 The **length** of a random file is limited by the capacity of the disk and the skill of the programmer. To get the most into a random file the programmer must plan to pack each block to capacity.

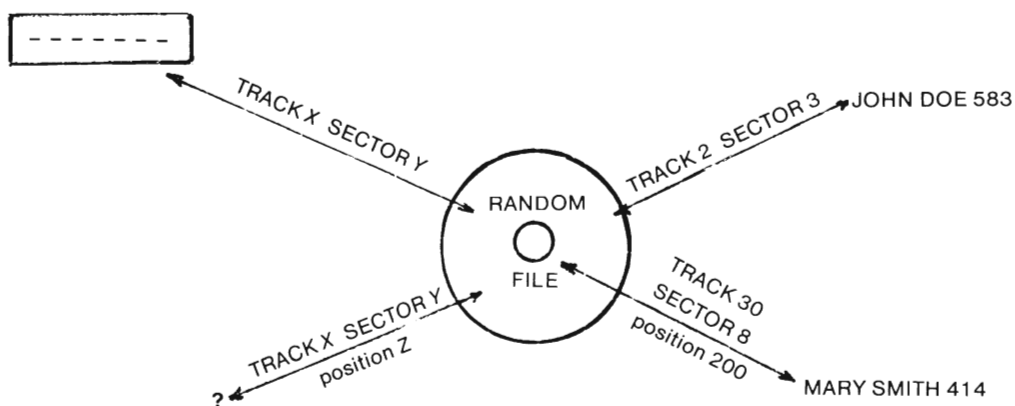
5123 The primary **advantage** of a random file is due to its nature. By specifying track numbers, sector numbers, and positioning a buffer pointer, any byte on any block can be accessed. Random files are especially good for disk related utility programs such as programs to make "back up" copies of disks.

5124 The disadvantage is that you get what you pay for. A great deal of bookkeeping must be done by the programmer. Two methods are usually used. An algorithm can be used to account for record numbers with their location (track, sector, position). A sequential file can also be used to store the 'keys' to the random file.

5125 A brief description of specific command formats and a sample program (of sorts) can be found in Chapter 8000.

(Continued)

5125 (Cont'd)



5130 Relative Files*

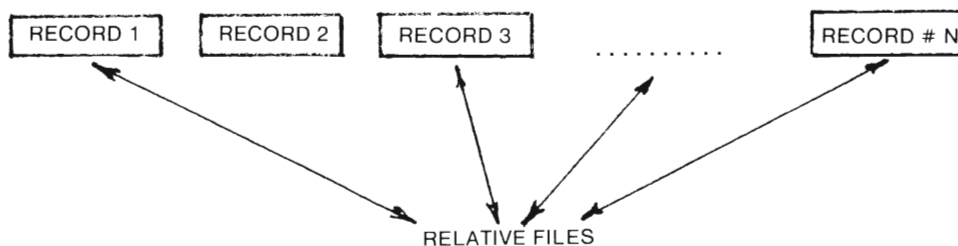
5131 The **nature** of a relative file is a random access file that is “relatively” easy-to-use. We can get at individual records and parts of those records.

5132 The **length** of a relative file is limited (but not very much) by a combination of factors including the length of each record and the number of available side sectors. (More details to follow). With relative files, we have a mere 160K bytes for our data. Part of a record can be on one block and the rest will be written to the next. i.e. We can span sections.

5133 The **primary advantage** of relative files is in their hybrid nature. They are only a bit more complicated to use than sequential files but still allow random access to individual records. The DOS does all the hard work for us. They are ideally suited to most data processing tasks. (Mailing lists, grade books, etc.)

5134 The **disadvantage** of relative files is again “you-get-what-you-pay-for.” The programmer must pay strict attention to file data format and record structure. (It’s not as hard as it sounds). The length of an individual record is restricted to a maximum of 254 bytes.

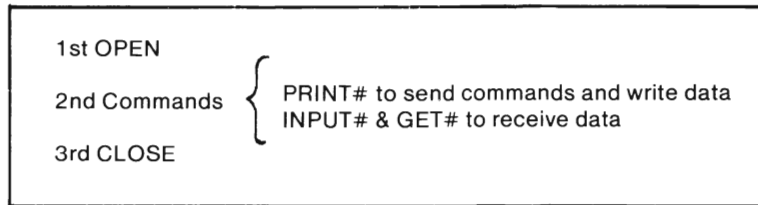
5135 A detailed discussion with specific command formats, operations, procedures, and sample programs can be found in Chapter 7000.



*These are the “cat’s pajamas”, neat, cool, etc. (No, I’m not that old.)

5200 COMMON PROCEDURES

Certain operations are common to all file types. Since the details vary considerably from type to type, this section is simply an overview. We are just trying to capture the essence of operation not a mastery of procedures.

5201 PROCEDURES OUTLINE: ALL FILE TYPES²

5202 All file work requires careful formatting of data. The more complicated the file type, the more attention is required. Essentially, things work like they do on the screen. What makes it a little tougher is that you can't "see" what's happening. (5400)

5203 An Error Channel Reading subroutine is included in all but the most elementary file using programs. The routines are similar to that discussed in the "Flashing Red Light" Chapter. It prevents the red light from flashing and protects your files when things go wrong. Some "error" conditions are actually necessary to successful operations. After you complete the next section, it would be a good idea to refer to the program in 9200 as a refresher.

5204 Channel Monitoring is required in advanced programs that combine file types and make use of other peripheral devices. The activities in this text don't cause problems in this area. We will use good programming practice by closing files as soon as possible after use.

5300 COMMANDS - General Format & Function

REM: Do not attempt mastery of this information at this time. Simply read and use as a reference later. If you did not take time for section 2300, you may want to do that as you go through OPEN and PRINT# here. (It's perfectly normal, I hope, to feel like you are chasing your tail.)

5310 OPEN

5311 FORMAT: **OPEN***fn,dv,ch,"message"*

5312 PARAMETERS:

fn = file number (range*: 1 to 127)

Technically 128 to 255 may also be used, but such is not recommended. Save these for printers that do not have an automatic line feed after each carriage return or when you want to double space on those that do. (Example in note 2331.)

dv = device number (range* : 8 to 11)

Use 8 for an unaltered drive. (160)

² - Even program "files" are handled this way. The DOS does it for you via channels 0 and 1 when you LOAD and SAVE programs.

* - Ranges specified are "inclusive." i.e. Both 1 and 127 may be used for *fn*, etc.

(Continued)

5312 (Cont'd)

ch = channel number (range* : 2 to 15)

15 is the command channel (also error channel). In disk drive operations, it is used strictly to send commands to the DOS and receive information from the drive itself, such as error messages.

2 to 14 are your data channels. They are used when reading from and writing to data files.

0 and 1 belong to the DOS. It uses them when SAVEing and LOADING programs. Although you can sometimes get by with using them, it's bad practice.

NOTE: ALL of the above must be specified in disk related operations.

"message" = STRING command to the drive (optional)

Various forms are used depending on the task at hand and the type of file in use. (See appropriate sections in chapters on specific file types.)

5313 FUNCTION: The OPEN statement "opens" a communication link between your computer and the drive (or any other peripheral device, for that matter).

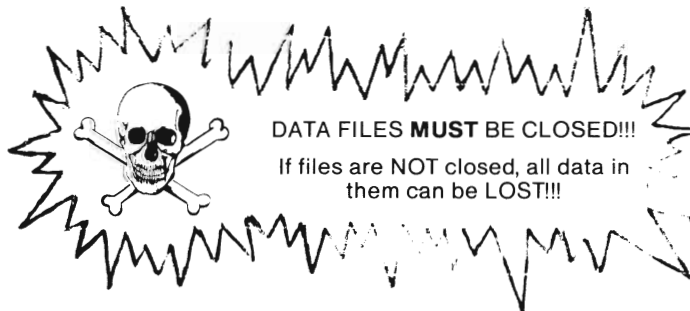
The file number is used by the system to keep track of the device number and channel so these need not be repeated in further statements (PRINT#, INPUT#, and GET#) until that file is closed. None of these statements will be "listened to" until the file is open. ("File not open error" results.)

Once a file has been opened, it stays open until specifically closed, either by the CLOSE command, 5320, ("File open error" results.) or another process.

5320 CLOSE

5321 FORMAT: **CLOSE***fn* *fn* = file number. See above.

5322 FUNCTION: If there is any data waiting in the buffer,² the CLOSE command causes it to be written to the file, and then 'closes' the link to the drive. In all write operations it is imperative that data files be closed by specific command. The INITIALIZE command (4300) may be used to close after read operations.



NOTE: Be sure to punctuate all commands as shown.

² - Think of a "buffer" as a 'holding pen' or a 'To Be Filed' basket. DOS will file the data when the buffer gets full or when told to do so by a command.

* - Ranges specified are "inclusive." i.e. Both 1 and 127 may be used for *fn*, etc.

5330 PRINT#

5331 **FORMAT:** **PRINT#** *fn*, *data file list*

or

PRINT# *fn*, *command string*

5332 **PARAMETER:**

fn = file number.

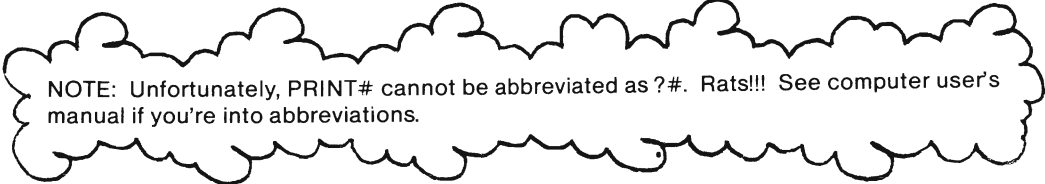
Each PRINT# statement refers to the file number, first number, specified in a previous OPEN statement.

5333 **FUNCTIONS:**

In general, the PRINT# statement directs output to a peripheral device, 2340. In disk drive operations, PRINT# is used to write data to files, first format above, or to send commands, second format above.

When the file number refers to a data channel (2-14), the "data file list" is structured like a normal PRINT to the screen. Variables (numeric and string), characters in quotes, and ASCII codes may be used. Punctuation works "normally". (See 5400 for details.)

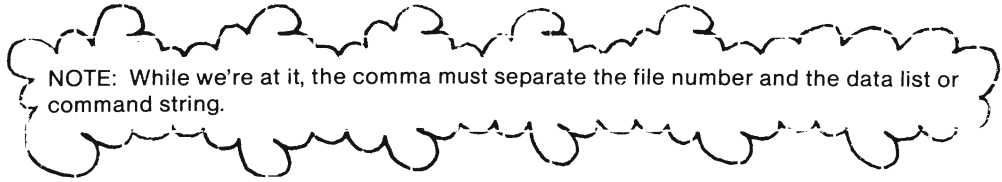
When the file number refers to the command channel (15), the "command string" is structured as directed for specific commands. General drive commands are covered in 4000. File commands are covered in chapters according to file type.



NOTE: Unfortunately, PRINT# cannot be abbreviated as ?#. Rats!!! See computer user's manual if you're into abbreviations.



NOTE: Using a space between PRINT and # will also result in a syntax error.



NOTE: While we're at it, the comma must separate the file number and the data list or command string.

5340 INPUT# and GET#

5341 **FORMAT:** **INPUT#** *fn, variable, variable, ..., variable*

and

GET# *fn, variable, variable, ..., variable*

5342 PARAMETER:

fn = file number

Each **INPUT#** and **GET#** statement also refers to the file number specified in a previous **OPEN** statement.

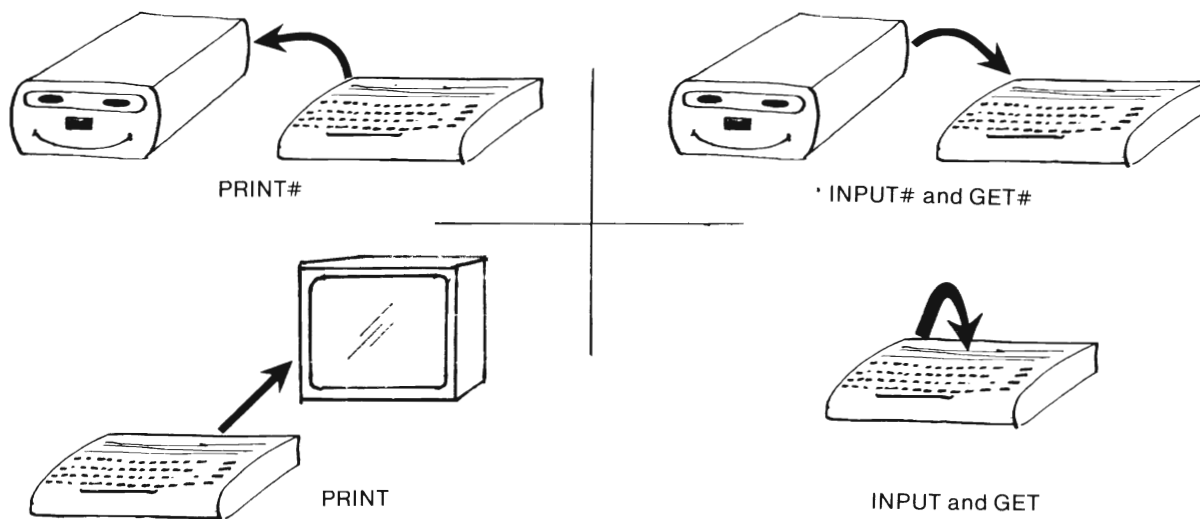
5343 The variable list may include any legal BASIC variables or combinations of them.

5344 FUNCTIONS:

Both statements are "input" statements. In disk drive operations both direct the system to receive information from the drive. Both function similarly to the keyboard commands **INPUT** and **GET**. Unlike **PRINT** and **PRINT#**, neither of these can be used in direct mode.

When the file number used in **INPUT#** refers to the drive, **INPUT#** will "fill" a variable until it "sees" a separator in the file. Care must be taken to co-ordinate **PRINT#** and **INPUT#** to insure successful file reading. (Details in next section.)

GET# gets data byte by byte from the file. Numeric variables use requires extreme care. As with **GET**, string variables can handle anything, numerics only numbers. (Details to follow.)

**5400 FILE DATA FORMAT**

REM: Writing data into a file is no big trick since **PRINT#** works exactly like **PRINT**. Things get a bit more interesting when we try to get the data out. **INPUT#** works just like **INPUT** and that's the rub. The **INPUT#** will fill the variable until it sees a separator, (comma or carriage return) in the file. The process is nearly automatic with **INPUT** in that the user types the "input" and hits **RETURN**. That's when **INPUT** sees the separator. For **INPUT#**, the separators must be placed in the file by the programmer.

This is best illustrated via examples. Again, mastery may not come on the first reading. The sequential file chapter does a lot more with this where you "fool around" with a sample program, 6260.

5401 **Example Set Up:** In all of the following examples, assume we've opened a fictitious file.
99 is the file number of that fictitious file.

We'll use: A\$ = "AAA" AND B\$ = "B 12"

Our goal: X\$ = "AAA" AND Y\$ = "B 12"

5410 **The Problem**

PRINT#99,A\$B\$

or puts AAA←B 12← into the file.

PRINT#99,A\$;B\$

This symbol will be used in this section to denote a carriage return.

NOTE: A carriage return is placed into the file automatically after every PRINT# statement.

NOTE: CHR\$(13) is ASCII for "carriage return".

NOTE: The arrow is not really a carriage return to the system. It is merely a teaching device.

5412 INPUT#99,X\$,Y\$ will fill X\$ = "AAA B 12"

Y\$ will be filled with whatever follows the carriage return in the file. That causes a lot of trouble if there is nothing there.

NOTE: INPUT# must be told when to stop filling the variable.

5413 Here's how to get the job done. The programmer has these options:

Use multiple PRINT# statements which will put the carriage return in automatically. (5420)

or

Put the separators into the data list in the PRINT# statement. (5430)

5420 **Multiple PRINT# Statements**

PRINT#99,A\$ } puts AAA←B 12← into the file
PRINT#99,B\$ }

the ← denotes the automatic carriage return after each PRINT#.

INPUT#99,X\$,Y\$ now fills X\$ = "AAA" , "sees" the carriage return

and fills Y\$ = "B 12" as desired.

5430 **Put Separators into the File**

5431 Set C\$ = CHR\$(13) (the ASCII code for carriage return)

PRINT#99,A#C#B\$ puts AAA+B 12+
 carriage return placed by C\$ automatic carriage return from PRINT#

Now INPUT#99,X\$,Y\$ fills the variables as in 5420 above.

X\$ = "AAA" AND Y\$ = "B 12" as desired.

5432 Set C\$ = CHR\$(44) (ASCII code for a comma)

or C\$ = "," (equivalent to above)

PRINT#99,A#C#B\$ puts AAA,B 12+ into the file
 "comma" placed by C\$ automatic carriage return

INPUT#99,X\$,Y\$ fills X\$ = "AAA" ,sees the "comma"
 and fills Y\$ = "B 12" as desired.

5440 **PRINT# Punctuation:**

Punctuation in the data list of a PRINT# statement has the same effect as it does in screen PRINT statements.

5441 ; separates variables in the data list. It is really only necessary to separate numeric variables. (See 5410.) If used at the end of a PRINT# data list, it will suppress the automatic carriage return.

PRINT#99,A#B\$; puts AAAB 12 into the file
 notice - no carriage return

INPUT#99,X\$,Y\$ will fill X\$ with the above and everything that follows until it "sees" a comma or a carriage return. That will result in a "string too long" BASIC error.

5442 , causes spaces in the file just as it does on the screen.

PRINT#99,A\$,B\$ puts AAA B 12+ into the file
 causes spaces automatic carriage return

INPUT#99,X\$,Y\$ fills X\$ = "AAA B 12"

NOTE: Commas are just dandy for screen formatting but, direct use of commas in PRINT# data lists, simply waste space and do not act as separators for INPUT# statements.

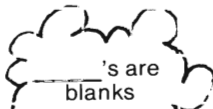
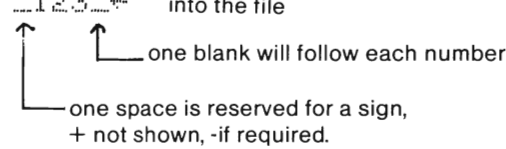
5450 **Numeric Variables**

PRINT#, INPUT#, and GET# statements can, of course, handle numeric variables. Care must be taken when mixing strings and numerics. Remember, when the system expects to find a number, it sends you an error message. Keyboard INPUT statements simply tell the user to "REDO FROM START". INPUT# is not that friendly.

5451 Example set up.

99 is again our fictitious file and "←" denotes a carriage return. (Remember ← is for sample purpose only and to the system is NOT a carriage return.)

let N = 123, N\$ = "123", and VN = VAL(N\$)

5452 PRINT#99,N puts `_123_←` into the file



5453 PRINT#99,VN also puts `_123_←` into the file, but

5454 PRINT#99,N\$ puts `123←` into the file

Notice the similarities in 5452 and 5453 and the difference with 5454. All of these can be read with either

5455 INPUT#99,X or INPUT#99,X\$





6000 SEQUENTIAL FILES

CHAPTER DIRECTORY

6100	PRELIMINARIES
6110	Overview
6120	OPEN to Write
6130	OPEN to Read
6140	Notes on Use
6150	Replace Option/Editing
6200	SAMPLE PROGRAM
6210	Objectives
6215	Preliminary User Instructions
6220	LISTING
6230	Explanation - Detailed
6240	User Instructions
6250	Experiments - Making Mistakes
6260	Exploring file data format
6300	COMMAND DEMO
6310	Objectives
6320	User Instructions
6330	Listing
6340	Explanation
6400	"A Can of Worms"
6410	Objectives
6420	User Instructions
6430	Listing
6440	Sample RUN
6450	After Thoughts
6460	"That's all she wrote"

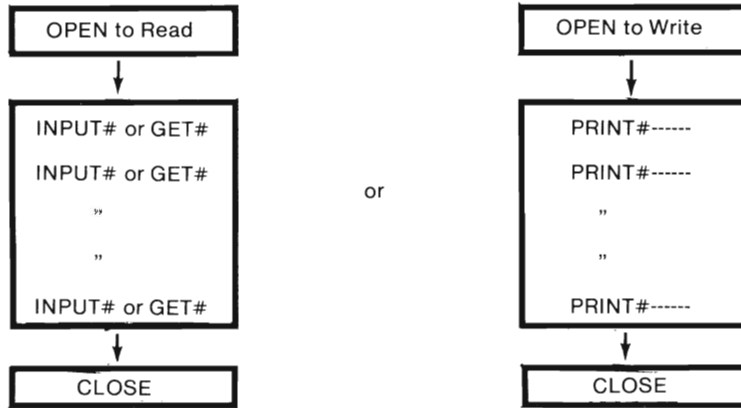
6100 PRELIMINARIES

Rem: If you are an "experience first" type or are simply up to your ear lobes with reading about, do (6200) and come back when ready.

6110 Overview

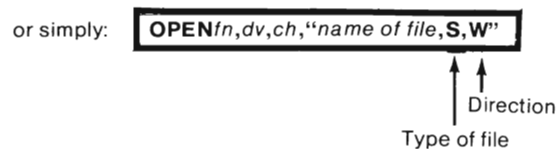
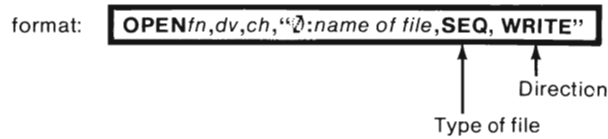
As with all files, we must OPEN the "drawer" before we can put anything in or get anything out. When we OPEN a sequential file we must:

- 1) Specify the exact name of the file.
- 2) Indicate the type of file (sequential, in this case).
- 3) Tell the system whether we want to read or to write. (Sorry but we cannot do both at the same time).



The file must be on the disk before we can read it and cannot already exist if we choose to write. (Do you see the analogies with LOAD and SAVE? As with SAVE, there is a replace option (6150) for re-writing existing files.

6120 OPEN SEQuential file to WRITE



6130 **OPEN SEquential file to READ**

format: **OPEN***fn,dv,ch,"name of file,SEQ,READ"*

Direction

or simply: **OPEN***fn,dv,ch,"name of file,S,R"*

6140 **Notes on Use**

6141 2-14 are used for *ch* = channel number. The other parameters, *fn* = file number, and *dv* = device number are described fully in (5310). Name specifications are the same as those for programs (3125).

6142 **PRINT***#fn,* may only follow an OPEN to WRITE.

INPUT*#fn,* and **GET***#fn,* may only follow an OPEN to READ.

6143 When the "WRITE" process is completed the file must be closed,

CLOSE*fn* or data can be lost.



6144 When the "WRITE" process is completed and the file closed, that file may be re-opened for READING via (6130). The parameters need not be the same.

6145 When a file is OPENed to WRITE, an error condition will result if the "name of the file" is in use, even as a program (PRG) or Relative (REL) file. (261)

6146 When a file is OPENed to READ, an error condition will result if the directory does not contain "name of the file" marked SEquential. (261)

6147 When the "READ" process is completed the file should be closed,

CLOSE*fn* so you don't run out of channels. This is not as critical as (6143) but it's bad practice to omit.

6148 If your program is interrupted by a SYNTAX ERROR use

CLOSE*fn* in the direct mode or data can be lost.

6150 **Replace Option**

Rem: Notice the similarity to the SAVE with REPLACE option (3230). The problems (3330) may also be present.

6151 Format: `OPEN/n,dv,ch,"@:file name,SEQ,WRITE"`

or simply: `OPEN/n,dv,ch,"@:file name,S,W"`

6152 Parameters are the same as described earlier (6141).

6153 Purpose: Allows new data to be written in place of an existing file. It is equivalent to SCRATCHing the existing file (4500) and then writing the new data in the ordinary manner.

6154 Edit Method 1: If the DIMension parameters permit, read the existing data into an array. Re-write with either the replace option above or a SCRATCH followed by a normal write.

6155 Edit Method 2: If all you need to do is add to (append) an existing file, use a combination of COPY (4800) and RENAME (4600).

Speculation: I no longer feel save using the replace option. Could it be that using the drive number, 0, will keep us out of occasional troubles?

Somebody's Law: If there is going to be a Replace Option foul up, an important file or program will be "replaced" by a "throw away test."

Check out 3360 if you haven't seen it yet. Using "0:" may not be as "optional" as I thought.

6200 SAMPLE PROGRAMS

6200 SAMPLE PROGRAMS

Rem: Experience is by far the best teacher (and usually the most difficult). By "playing" with this program, you'll gain a feeling for how all of this fits together. That's the primary objective of this section.

6210 PROGRAM OBJECTIVES

This program will allow the user to
 WRITE a sequential file with PRINT#
 READ that file via INPUT#
 "GET" a computer's eye view of that file via GET#
 CHANGE that entire file
 QUIT the program.

The program also demonstrates the technique of "reading the error (command) channel" via a subroutine.

6215 **Preliminary User Instructions²**

1st: ACTIVATE your WEDGE (1230). You can do most of this section without but...

2nd: Load the program from "FRIENDLY FLOPPY"

 Name: 1ST SEQ FILE PGM

3rd: If you want to try it before studying the explanation (6230) have fun. GOTO 6240

4th: As you study (6230) you may want to list the program by section:

LIST - 100:	Preliminary Set up.
LIST 300 - 399:	WRITES
LIST 400 - 499:	Changes
LIST 600 - 699:	READ
LIST 800 - 899:	"GETS"
LIST 9999 - :	Error Sub routine

6220 **LISTING of "1st SEQ FILE PGM"**

Rem: Just in case something happens to "FRIENDLY FLOPPY."

```

10 REM *** 1ST SEQ FILE PGM ***
20 OPEN15,8,15
21 C#=CHR$(13)
22 M#=", "
25 PRINT"Q":INPUT"FILE NAME";FF#
30 PRINT"Q"FF#C#C#"READ, WRITE, GET, CHANGE, OR QUIT"
40 GETK#:IFK#=""THEN40
45 IFK#="Q"THEN10001
50 IFK#="R"THEN600
55 IFK#="G"THEN300
60 IFK#="C"THEN400
65 IFK#<"W"THENPRINT"R,W,G,C,Q";GOTO40
300 OPEN3,8,3,FF#+",S,W"
310 PRINT"TO WRITE: "FF#C#
320 INPUT"NAME";N#
330 INPUT"TEST SCORE";T#
340 PRINT#3,N#M#T#
341 GOSUB9999
350 PRINT"ANOTHER?(Y/N)"
355 GETK#:IFK#=""THEN355
360 IFK#="N"THENPRINT#3,"*":CLOSE3;GOTO30
370 IFK#<"Y"THEN350
380 GOTO320
400 PRINT"TO CHANGE: "FF#C#
405 PRINT"SHOULD Q"FF#" BE SCRATCHED??(Y/N)"
410 GETK#:IFK#=""THEN410
415 IFK#="N"THEN25
420 IFK#<"Y"THEN410
430 PRINT#15,"S0:"FF#
431 GOSUB9999
440 GOTO300
600 OPEN3,8,3,FF#+",S,R"
610 PRINT"READING FILE: "FF#C#C#
620 INPUT#3,N#,T#

```

(Continued)

² - Hopefully by now you can do all the "check lists." If you've forgotten anything see 1010. Pre load check list.

6220 (Cont'd)

```

621 GOSUB9999
625 IFLEFT$(N$,1)="*"THENCLOSE3:GOTO30
630 PRINTN$TAB(25)T$
640 GOTO620
800 OPEN3:8:3:FF$+";S,R"
810 PRINT"GET:  "FF$C$
820 GET#3,G$
821 GOSUB9999
825 IFG$=CHR$(13)THENG$="←"
830 PRINTG$;
835 IFG$<"*"THEN820
840 PRINTC$
841 CLOSE3
842 GOTO30
9999 INPUT#15,EN,EM$,ET,ES
10000 IFEN<20THENRETURN
10001 PRINTEN;EM$,ET;ES:CLOSE3:CLOSE15:END

```

6230 **SAMPLE PROGRAM - EXPLANATIONS**

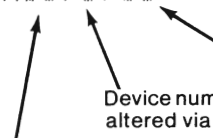
Rem: Explanation for using BASIC statements have been deleted but their function described. Consult a BASIC reference for more details. No "crunching" has been done to the program to make it easier to follow. In this section all "line numbers" references refer to the program. Numbers enclosed in "()" refer to text section.

6231 **Initial Set Up (LIST - 100)**

```
10 REM *** 1ST SEQ FILE PGM ***
```

This REM simply contains the Exact Name of the Program. I started doing this when I started forgetting which program was which.

```
20 OPEN15,8,15
```



 Command channel. Must be 15.

 Device number. 8 for unaltered drive. (If by any chance your drive has been altered via hardware, use that new number here.) (160)

 File number. In this program it is matched to the channel number. If changed, corresponding change has to be made in lines 430, 9999 and 10001.

This line opens the command channel so we can read any error message from the drive and protect files.

```
21 C$=CHR$(13)
```

sets a simple string variable equal to a carriage return using ASCII code number 13.

It is primarily for use as a "separator" for our file data format (5400) experiments in (6260). It has also been used in normal screen PRINT statements.

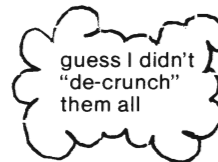
6231 (Cont'd)

22 M\$="," sets a string variable equal to a "comma." It will serve as a "separator" (5400).
 M\$=CHR\$(44) is an equivalent statement.

NOTE: Lines 20, 21 and 22 were originally "crunched" into one line as were other lines numbered consecutively.

```
25 PRINT "C"; INPUT "FILE NAME"; FF$
```

Here we just clear the screen and ask the user to input the "file name."



Note: FN\$ was my first choice but I got a syntax error. The system thought I wanted "FUNCTION" in BASIC.

Note: If you use something other than FF\$, you of course, will have to make corresponding changes in lines 300, 400, 600, 800, 30, 310, 410, 610 and 810.

```
30 PRINT "C"FF$C$C$ "READ, WRITE, GET, CHANGE, OR QUIT"
```

Line 30 is the "job menu." The program will come back here whenever the current task is finished. "Cursor down" in quotes would do the same thing as C\$ here since the cursor is already at the far left.

```
40 GETK$: IFK$="" THEN40
```

waits for the user to hit a key.

```
45 IFK$="Q" THEN10001
```

```
50 IFK$="R" THEN600
```

```
55 IFK$="G" THEN800
```

```
60 IFK$="C" THEN400
```

} each sends the system to the line in the program that will do the job. If the user types the entire word—no problem.

```
65 IFK$<>"W" THENPRINT "R,W,G,C,Q": GOTO40
```

The line above is a "user friendly" line so the program does not default to "write" and even gives the user a hint.

NOTE: Lines 45-65 are prime candidates for ON ____ GOTO ____, ____, ... Doing so would require changes in the job menu structure. Asking the user to input a number from 1 to 5 would be the easiest. ("GEN REL," the relative program sample illustrates this technique(7316).)

6232 FILE WRITE (LIST 300 - 399)

```
300 OPEN3,8,3,FF$+"",S,W"
```

↑ ↑ ↑ ↑ ↑ ↑ ↑

File number. Again I've chosen to match the channel number. Changes here would demand changes in 360 and 10001 which would effect all other OPEN and CLOSE 3 statements.

Device number (see line 20)

Channel number. Data channel 2 to 14.

string holding file name

"adds" strings

file type: SEQUENTIAL (6120)

direction: WRITE (6120)

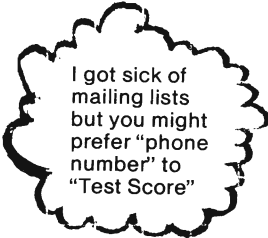
Note: This line OPENS the sequential file to write. If FF\$ already exists the poor user is in trouble as you'll find out in "making mistakes." (6250)

```
310 PRINT"TO WRITE: "FF$C$ is just a message to the user.
```

```
320 INPUT"NAME":N$
330 INPUT"TEST SCORE":T$
```

Both of these simply ask for user to enter data.

Note: String variables were used to avoid difficulty. We'll mess with that later.



```
340 PRINT#3,N$M$T$
```

↑ ↑

File number. Must agree with the first number in line 300.

There's the separator.

"writes" data to the file. (Well, actually to a buffer and then to the disk). None of the data will be useful until the file is CLOSED.

NOTE: This statement will put 1st user input,2nd user input ↵ into the file. Again, " ↵ " is used to show a carriage return. What is really "in the file" is the byte that signifies CHR\$(13).

```
341 GOSUB9999 sends the system to the error channel read routine. See line 9999 for details.
```

Note: One of these probably should follow line 300 to catch an already existing file earlier in the program. (6250) Making mistakes will illustrate.

```
350 PRINT"ANOTHER?(Y/N)" Just asks the user if there will be another entry,
```

```
355 GETK$:IFK$=""THEN355 and waits for a response.
```

6232 (Cont'd)

```

350 IFK#="N" THEN PRINT#3, "*" : CLOSE#3 : GOTO#30
    
```

must agree with first number in line 300.

┌──────────────────┬──────────────────┬──────────────────┐
 │ Puts a "flag" (and an automatic carriage return) into the file. The * will be used to stop the file reading processes. │ │ To job menu for next task. │
 └──────────────────┴──────────────────┴──────────────────┘

Causes the system to put the stuff in the buffer onto the disk and updates BAM (270). If this is not done, all the data the user just typed is inaccessible!

```

370 IFK#<>"Y" THEN 350
    
```

is just a user protector. If omitted any key except "N" would send the system back for another entry.

```

380 GOTO#320
    
```

line 320 is the user input line.

NOTE to the more advanced.
 The STATUS function can be used to determine the end of file eliminating the need for this "flag" system. (6344, 6347, 8330, 8500)

6233 FILE CHANGE (LIST 400 - 440)

```

400 PRINT"TO CHANGE: "FF#C#
405 PRINT"SHOULD I"FF#" BE SCRATCHED??(Y/N)
410 GETK#:IFK#="" THEN 410
415 IFK#="N" THEN 25
420 IFK#<>"Y" THEN 410
    
```

The lines above allow the user to abort scratching a file. 400 and 405 are just messages. 410 is the usual "wait for user." 420 just protects the user. i.e. The Y-key must be pressed to scratch the file. In some programs a line like 420 can be omitted if the consequences of user error are not severe.

```

430 PRINT#15, "S0:"FF#
    
```

is the actual command to SCRATCH (4500) the file.

more commands
 in 6300

notice that we address the command channel not the data channel. The 15 here is the first number (file number) from line 20.

```

431 GOSUB#9999
    
```

← the error channel read routine.

```

440 GOTO#300
    
```

← the beginning of file write (6232)

Note: The original version of this program used the replace option. I discarded it when I heard of and encountered problems with it. So sad. (6150)

6234 FILE READ (LIST 600 - 640)

600 OPEN3,6,3:FF#+",S,R"

↑
direction:READ(6130)

Note: Other parameters are identical to those in line 300. The file number and channel numbers need not be the same as those used in 300 but doing so simplifies matters.

610 PRINT"READING FILE: "FF#C#C# is just a message to the user like lines 310 and 410.

620 INPUT#3,N\$,T\$

↑
must agree with file
(1st) number in 600.

↑
commas must be used in the INPUT# statement no matter what "separators" are used in the PRINT#. We'll experiment in (6260).

621 GOSUB9999 again sends the system to read the error (command) channel.

625 IFLEFT\$(N\$,1)="*"THENCLOSE3:GOTO300

back to job menu

not as critical as that in line 360 to data, but if omitted from program other OPEN statements will cause BASIC syntax errors. If the program had another WRITE file open, its data could be lost unless it is CLOSED

↑
our stop
flag placed
by line 360

This part is a hold over from a previous version of this program. (I was going to fix it to simply

IFN\$="" but it does demonstrate a user protection technique. (See basic BASIC reference).

Also (7332) line 2055.

630 PRINTN\$TAB(25)T\$ will display data on screen.

Note: Feel free to modify this line if you don't like the display format.

640 GOTO620 sends the system back for another entry. It keeps up the cycle until it "sees" the flag (*).

Note: If something happened when the file was written to prevent the writing of *, we get into an "eternal" loop. **RUN/STOP** will have to be used and files closed by CLOSE3 or INITIALIZE (4300). (6250) for examples.

6235

FILE "GET" (LIST 800 - 842)

Rem: This sequence, although intended to enable you to explore file data format, could be adapted to explore the contents of an "unknown" file. It will return punctuation used as separators as well as carriage returns.

800 OPEN3,8,3,FF#+" .S.R" is identical to line 600 in both structure and function.

Note: Here again the parameters, "3,dv,3" need not be the same as previous open statements, but . . .

810 PRINT"GET: "FF#C\$ is just a message to the user. (C\$ is still just a carriage return).

820 GET#3,C\$ will "get one byte (character) from file 3. The next time through this line will get the next character, etc. As you will see GET # is a slow way to read the file but it will "get" through the file no matter what—assuming we ask to "get" a string.

↑
must agree
with the file
(1st) number
in 800

821 GOSUB9999 reads the error channel (again).

825 IF6#CHR\$(13)THENC\$="↵"

⏟
a carriage return
(placed automatically
by line 340)

↑
you'll see these when you run the GET part of the program.

NOTE: This is a line to help you "see" what the system "sees."

830 PRINTC\$: places what the system found where you can see it. The semi-colon was used in the normal manner.

835 IF6#C\$*" THEN820 stop flag again. (The file does contain one more carriage return).

840 PRINTC\$ is just 2 carriage returns, one from PRINT and one from C\$.

841 CLOSE3 must use file number from 800. See also line 625.

842 GOTO30 and back to the menu for another task.

6236

ERROR CHANNEL SUBROUTINE

```
9999 INPUT#15,EM$,ET,ES
```

Must be the file (1st) number used in the OPEN statement in line 20.

Error location. ET = Track
ES = Sector

Any basic variables can be used. Line 10001 must be changed accordingly.

Error message STRING. May be any string variable.

Error number. Numerical variable is recommended. If a string is used here a VAL function must be used in 10000.

Note: Commas required in INPUT# statements.

```
10000 IF EN<20 THEN RETURN
```

Here is where the error read subroutine starts to differ from the program listed below and described in (9200). Error numbers less than 20 do not indicate error conditions. If all is well, we go back to the main part of the program and continue the task.

```
10001 PRINTEN:EM$:FT:ES:CLOSE3:CLOSE15:END
```

Simply prints message to screen. Feel free to adjust the format.

usual meaning
closes command channel.

Closes data channel. Here's where we'd have a mess if we didn't use the same file numbers in all data file open statements (300, 400, 600, 800).

Note: Our "Quit" option ends up here. If you don't want to use the "OK" error message use a line 10002 for COSE3:CLOSE15:END and fix line 45.

Note: Program from (9200) for your convenience.

```
60000 REM***READ COMMAND/ERROR CHANNEL
60010 OPEN15,8,15
60020 INPUT#15,EM$,T,S
60030 PRINT:EM$,T,S
60040 CLOSE15
```

for comparison only

6240

USER INSTRUCTIONS

Activate your wedge—just in case (1230). You can work without it, but ...

LOAD "1ST SEQ FILE PGM",8

NOTE: "TEST 1" is on "FRIENDLY FLOPPY"

DO NOT name your file "TEST 1" if "FRIENDLY FLOPPY" is in the drive.

DO NOT SCRATCH TEST 1

SUGGESTION: Change disks after loading the program. INITIALIZE (4300) before running. @ I

6241 Type: RUN

The screen will clear and the programs will ask for a file name.

We'll write a test file. Take care NOT to choose the name of a file on the disk in the drive. If you do so, the red light will flash. GOTO 6251

Type: _____ 

The screen will show:

file name
READ, WRITE, CHANGE, GET, or QUIT

You can now select a job by typing the word or just the first letter. See lines 30 - 65 in the program explanation. (6231)


6242 Type: W

We have to WRITE the file before we can read it or do anything with it except QUIT the program.

The screen will show:


TO WRITE: file name
NAME?

It wants the name for this record.

Type:  Anything you want. Make up your own name but keep it under 20 characters or you'll mess up the display when we read the file.

The screen will add:

TEST SCORE?

Type:  Anything you want—but for later uses, give it a number like 100.

When the screen shows:

ANOTHER(Y/N)

Type: Y

Write 2 or 3 more records. Make up your own names but use a negative number for one test score, a positive for another, and one with a decimal point, like 6.2, for a third. DO NOT use any letters even though this program can handle them.

When you are finished entering your records,

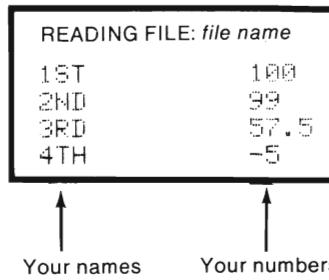
Type: N

You'll see the "job menu" again. This time we'll read our file.

² - Be sure to use a formatted disk (2000).

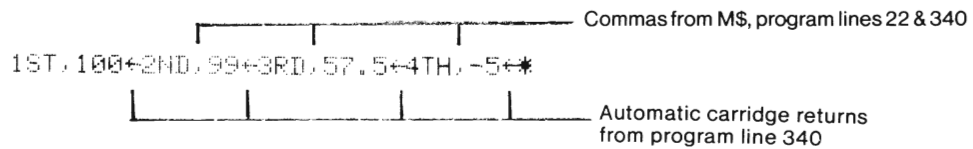
6243 Type: R

The screen will show:



When the menu appears, we'll "get" a look at the file the way the computer "sees" it.

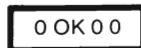
6244 Type: G The characters will come on, one at a time.



When you see the menu, if you want to rewrite the entire file, type C for change. (It is a nuisance. We'll have a better way in a different program). To escape the program...

6245 Type: Q

The screen shows:
That's just the
error channel read
out from line 10001. (9250)

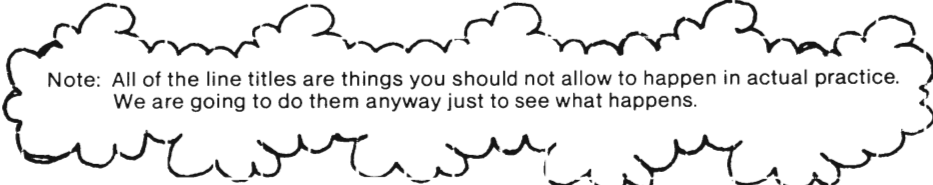
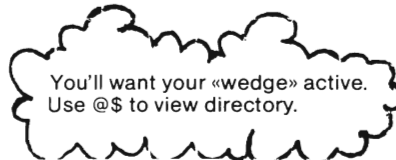


6250 **Making Mistakes**

Rem: A "user friendly" program does not permit the user to mess things up too terribly. This program is far from it as you will see. The kinder the program, the longer it gets. Take a look at COMMAND DEMO in 6330 if you haven't already done so.

As you work through this section I'll ask you to make mistakes. Hopefully you'll learn how to get out of trouble and how to avoid it in your own programs.

Before we go on, view the Directory of "Friendly Floppy." Be sure you see a file called "TEST 1" If you don't, write one (6242).



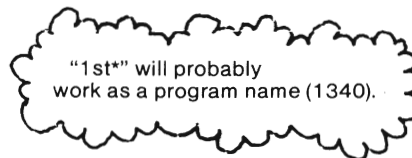
6251 OPEN an EXISTING SEQuential file to WRITE



Rem: That's exactly what we are going to do.

Activate your «wedge»

Load and run: 1ST SEQ FILE PGM



Ask for: TEST 1 or any other existing sequential file.

Type: `W` The red light should be flashing. (Just relax and continue).

Type: A name and a number or some gibberish, or simply hit RETURN a couple of times.

The red light goes out. You see the error message and the program ends. Bad practice. The user should not see the red light unless there is a real problem (like hardware failure).

Type: `301 GOSUB9999` This adds another trip to the error channel.

Repeat this experiment with the new line. See the difference? If the programmer wants to be really kind, error handling routines can be built to look for specific error messages. User options can be built accordingly. More sophisticated routines are used in COMMAND DEMO (6300).

NOTE: It is good practice to read the error channel after OPEN statements.

6252 OPEN a NON EXISTENT SEQuential file to READ



Rem: This works just about the same way as the preceeding.

RUN the program again and this time use a name that does not exist on the disk, say TEST 2 (unless, of course, you've already written one).

Ask to read, Type: `R`

Notice there is no flashing light this time. The program hit the error channel read routine very quickly. The error message appears (due to line 621). It is still good practice to add an error channel read routine immediately following an OPEN statement.

Fix that by adding `601 GOSUB9999`

6253

User interrupts run.

A matter of taste. Sometimes a good idea to disable RUN/STOP key.

Run the program, ask for TEST 2

or
any other file NOT on the disk and the
"WRITE" option. Start writing the file.

One or two records will be enough. When the programs asks for ANOTHER (Y/N)
hit the RUN/STOP key.

Notice that the break message appears but that the red light is "on solid" (0146). View your directory. (If your «wedge» is not active, @\$ doesn't work, load the directory (1218). You'll of course have to reload the program (1220)).

0	DISKNAME	ID	2A
1	TEST 1	SEQ	
0	TEST 2	*SEQ	

The * in front of the file type shows that the file was NOT CLOSED. For all practical purposes the data is unavailable.

Run again. Ask for TEST 2


to Read or Get. The red light goes out but we see yet another error message.

60 - READ FILE OPEN

Run again. Ask for TEST 2 to Change.

Allow the system to scratch. When the WRITE portion appears, repeat the experiment, but...i.e.

Write an entry
Hit RUN/STOP when you see ANOTHER(Y/N)
View your directory (notice the *SEQ)



@\$ to view
directory with
active wedge

This time

Type: CLOSE3

The red light goes out. (The drive may "whir".)

View the directory again. The * in front of SEQ is gone.

The file is now closed. This program will not be able to read it properly but at least you can "get" at your data.

Try one more run, this time with TEST 2 to read. When you get sick of seeing your last entry, use the RUN/

STOP key. You might as well SCRATCH (4500) TEST 2 while you're thinking about it.

(Continued)

6253 (Cont'd)

Note: If the user in a BASIC error condition interrupts a file write operation, CLOSE the files manually if at all possible. You need to know the file number.

`CLOSE/n`

Some programmers under some circumstances like to disable the RUN/STOP, RESTORE key for at least parts of the program run. These Pokes will do the job².

POKE808,225 to disable RUN/STOP, RESTORE Keys

POKE808,237 to (re)-enable

Frankly, I'm not in the habit of using this option. Then just POKE to disable can be a real hazzard when you are in a debugging stage of a program. If the RUN/STOP is disabled and you're in trouble, remove the disk, power off the computer and start over, (not a nice prospect if you haven't SAVED for a while)

6260 **EXPLORING FILE DATA FORMAT**

Rem: If your goal is to simply use file programs, you can skip this section. On the other hand, if you want to write your own programs, you should spend some time here. The methods also apply to RELATIVE and to RANDOM files.

6261 **STRING VS NUMERIC**

The 1ST SEQ FILE PGM writes and reads all data with strings.

The SEQ FILE # PGM writes and reads the test score data with numeric variables. The programs are identical except for these lines:

```
330 INPUT"TEST SCORE";T
340 PRINT#3,N#M#T
620 INPUT#3,N#T
630 PRINTN#TAB(25)T
```

All that's different is the numeric variable, T, replacing the string variable, T\$.

Load the SEQ FILE # PGM, run, and write a file just as you did in (6242). Read your file. It will look like the one below. I called mine "TEST NUMERIC."

```
TEST NUMERIC
1ST            100
2ND            99
3RD            67.5
4TH            -4
```

Do you see a very subtle difference from TEST 1? Hint: Look at the negative number entry(6243). The"—" is in a different location. These numeric entries save room for the sign, + not printed.

Try the "GET"

```
TEST NUMERIC
1ST, 100 +2ND, 99 +3RD, 67.5 +4TH, -4 +# Compare to that in (6244).
```

(Continued)

² - COMPUTE!'s GAZETTE, September 1983, Page 14

6261 (Cont'd)

The changes here are not nearly so subtle. Notice the space after each numeric entry. Reread (5450). Read and Get "TEST 1" with this program. (2nd SEQ FILE PGM).

```
TEST 1
1ST      100
2ND      99
3RD      57.5
4TH      -5
```

Test 1, written with T\$ but read with T. We'd be in big trouble if any of the T\$'s had contained non-numeric data. The "GET" part is identical. The content of "TEST 1" have not changed.

```
TEST 1
1ST, 100 +2ND, 99 +3RD, 57.5 +4TH, -5 +*
```

Reload: 1ST SEQ FILE PGM

Now we'll read the NUMERIC file with the string program.

```
TEST NUMERIC
1ST      100
2ND      99
3RD      67.5
4TH      -4
```

Notice that the numeric file reads like the string file (6243) when read with a string program. The "GET" job returns exactly what's in the file.

```
TEST NUMERIC
1ST, 100 +2ND, 99 +3RD, 67.5 +4TH, -4 +*
```

The two programs are nearly interchangeable so long as the numeric program finds only numbers for numeric variables. If it should find a string, a BASIC error condition will abort the run. The user must close the files by number if known or with the "INITIALIZE" command given via wedge @I or via PRINT#fn,"I" where fn is the file number open to the command channel. See (4300).

6262

Rem: **SEPARATORS**

Rem: Separators are described in 5400, the "Rem" and especially in 5430. All of the following use the string program 1ST SEQ FILE PGM. We'll "play around" with separators. All of these experiments will involve altering the program. To get these to work as described, be sure to follow directions. You can, of course, make up your own experiments.

Recall: 22 M\$="," is used to place separators into the file in 340

```
340 PRINT#3, M$M$T$
```

These are the lines that we will be changing.

6263

CARRIDGE RETURN

Type: LIST340 and change it to read as shown below:

```
340 PRINT#3,N#C#T#
```

↑
Change M to C

Run the new program and write a sample file called "TEST CR."
Read and Get jobs will look something like this:

```
TEST CR
1ST          100
2ND          99
3RD          45.4
4TH          -8
```

Notice that the read process results are the same as those with the comma used as a separator.

```
TEST CR
1ST+100+2ND+99+3RD+45.4+4TH+-8+*
```

↑ ↑ ↑ ↑ ↑ ↑

CS CS CS CS

```
TEST 1
1ST,100+2ND,99+3RD,57.5+4TH,-5+*
```

↑ ↑ ↑ ↑ ↑

MS MS MS MS

Automatic after 340

You'll see the difference when you use the "GET." Compare "TEST 1" with "TEST CR."

As a programmer, the choice is yours. All of the following versions of 340 will be read the same way.

```
340 PRINT#3,N#C#T#            where C$=CHR$(13)
```

```
340 PRINT#3,N#M#T#            where M$=CHR$(44) or M$=","
```

```
340 PRINT#3,N#"", "T#        note the quotes!!!!
```

6264

'REAL' COMMA

Type: LIST340 and change it to read as shown below:

```
340 PRINT#3,N#"", "T#
```

Important

Run this new program, write a sample file with 4 entries called "TEST REAL COMMA." This time, use the "GET" before you read!

(Continued)

6264 (Cont'd)

Your results from the "GET" will look like this:

```

TEST REAL COMMA
1ST          100+2ND          99+3RD          Notice all the spaces. Notice that
              89.2+4TH          -1+*          there are only 4 separators!

```

Now read your file. The system thought "1st.....100" was the name, N\$, and "2nd.....99" was the Test Score, T\$.

```

TEST REAL COMMA
1ST          100          2ND          99
3RD          89.2          4TH          -1
              ↑
              screen spacing slightly different?

```

Now use the "CHANGE" to rewrite "TEST CR" so it contains 3 entries. After writing use the "GET" first and you'll see something like this. It will "scroll" on your screen.

```

TEST REAL COMMA
1ST          100+2ND          99+3RD          -99+*

```

Warning - be ready to hit RUN/STOP key after you ask to read this file.

```

TEST REAL COMMA
1ST          100          2ND          99
3RD          -99          *
3RD          -99          *
3RD          -99          *
3RD          -99          *
3RD          -99          *
3RD          -99          *
3RD          -99          *

```

Whew! Notice your red light is on solid.

Type: CLOSE3

The flag was not found in the proper position. Since the separators did not occur after the name, the flag was in a Test Score slot.

² - This space is due to a difference in the way the printer and the screen handle commas. You would not see it if I'd used the same number of characters for 1st and 3rd Test entries.

6265 SEMI COLONS

Type: LIST340 and change it back to its original form.

```
340 PRINT#3,N#M#T#
```

Type: LIST22 and change it to 22 M#=";"



Run the new program and write a file called "TEST SEMI." If you write an even number of entries, it will "read" in a "semi normal" manner. An odd number of entries will cause the same problem as in (6263). To avoid difficulties use the "GET" before you read.

```
TEST SEMI
1ST;100+2ND;99+3RD;-5.4+*
```

Automatic from 340

```
TEST SEMI
1ST;100          2ND;99
3RD;-5.4        *
3RD;-5.4        *
3RD;-5.4        *
3RD;-5.4        *
3RD;-5.4        *
3RD;-5.4        *
```

(with odd number of entries)² Notice that the semi colon is NOT recognized by INPUT# as a data separator. Semi colons do separate variables in PRINT# statements as they do with PRINT to screen.

Type: LIST340 and change M\$ to a real semi colon.

```
340 PRINT#3,N#;T#
```

Run the new program and write a file, "TEST REAL SEMI." Even and odd numbers of entries will result as earlier. The "GET" job will look like this:

```
TEST REAL SEMI
1ST100+2ND99+3RD-88+*
```

```
TEST REAL SEMI2
1ST100+2ND99+*
```

Notice that your data is bunched the same way as if 340 contained no punctuation between N\$ and T\$. When you finish with SEMI, try the whole process with: 340 PRINT#3,N#T#

(Continued)

² - Type: CLOSE3 if necessary to shut off solid red light. Nothing awful will happen if you don't bother.

6265 (Cont'd)

These are your "READ" results with odd and even numbers of data entries.

TEST REAL SEMI2		TEST REAL SEMI	
1ST100	2ND99	1ST100	2ND99
*		3RD-88	*
		3RD-88	*
		3RD-88	*
		3RD-88	*
		3RD-88	*

6266 **COLONS**

Type: LIST340 and put it back to its original form.

```
340 PRINT#3,N#M#T#
```

Type: LIST22 and change it to 22 M#=":"

Run this new (and last) program. Write a sample file called "TEST COLON." Even and odd entries will have the similar effects as described earlier but another effect will also be apparent.

Use the "GET" first.

TEST COLON	
1ST:100+2ND:99+3RD:4.7+*	appears as anticipated.

And now the "READ." (This one for odd number)

TEST COLON		
1ST	2ND	
3RD	*	The system ignored the data after the colon. If you have a devious mind you might see a way to hide data in a file. Hmmm!!!
3RD	*	
3RD	*	
3RD	*	
3RD	*	

6267 The bottom line.

String programs are safer than numerics and save space in files.

Use the VAL function to work with the numbers.

Carridge returns and commas are the safest separators.

Carridge returns automatic after each PRINT# so maybe multiple print statements ain't so bad after all. (5420)

6300 Command Demo

Rem: You'll find this section useful if your intent is to write your own program. If you are simply a casual user you'll pick up a feeling for file management by at least using the program.

6310 Program objectives:

The primary purpose of **COMMAND DEMO** is to illustrate:

- the use of some of the commands that can be sent to the drive.
- alternate error channel read routines
- provide a "directory view" within a program.
- how a program can be written to be kind to the user.

6320 User Instructions

6321 Name: **COMMAND DEMO**

load type: (normal)

**6322 PRACTICE DISK -**

Be very careful that the disk you select for practice does not contain any files or programs that "match" any of the following: (4522)

FILE ? COMBO ? HOLD

↙ ↘

wild cards

For best results, "Format" a blank disk (2000) for this run. The "directory view" routine used in this program is written in BASIC so it is much slower than the machine language routine of your «wedge». A long directory will take a while to view and may also "bury" some of the changes where they will be hard to find. i.e. The files we are going to create are short so DOS is likely to stick them wherever it finds room. The program will still work but it will be harder to see the results.

6323 Load and Run the program.

You'll see the Title followed by a short form of the directions above. (6322)

Insert your practice disk, hit any key and the drive will INITIALIZE your practice disk (6343). You'll see the current disk directory. It will look a bit different than what you usually see. If you don't like your choice of practice disk, use the "left arrow" key to quit the program. If you chose to quit simply re-run the program to start over.

6324 **File Write and Read**

If you hit return, you find that the program is asking for data. It will want two pieces of data for each of four test files. I suggest you use something that shows order like 1, 2, 3, ... or A, B, C, ... The program will place your two entries and then a string of its' own actually (only the file name) into the file. It will come back to you for two more entries for the next file.

If by any chance you see an error message, check out your manual, or (9500). If it is a 63 FILE EXISTS, use your wedge to examine the directory carefully. If you find FILE 1, FILE 2, FILE 3 or FILE 4, that's the problem. Change disks or scratch (4500) those files (sure they are just practice files), and start over.

When you finish writing you'll see the directory. After you hit a key the program will simply read back the files you wrote.

Notice: Writing a file places the file name (and a bunch of other stuff 0270) into the directory. Reading a file does not affect the directory.

6325 **File Communication - The COPY command (4800)**

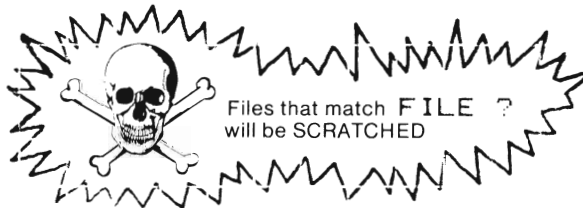
Sit back and relax. The program will 1st combine FILE 2 and FILE 4 into COMBO 1. If you are seeing a read-out of COMBO 1, it's already done. Notice also that COMBO 1 has been added to your directory. If you are thinking about programming, you can see the possibilities. This is a much better way to update files if all we have to do is combine some files.

COMBO 2 consisting of FILE 1, FILE 2, and FILE 3 will be formed in the same way. You'll see a reading of COMBO 2 and the directory (again).

6326 **RENAME a file (4600)**

This phase simply changes the name of COMBO 1 into HOLD. The contents of HOLD as you can see are exactly those of COMBO 1. HOLD is not a new file at all. Notice that COMBO 1 no longer appears in the directory. In its place, you should see HOLD.

If you got a 63-FILE EXISTS error, somebody probably ran this program but forgot to SCRATCH HOLD (4500).

6327 **SCRATCH FILE ? (4500)**

Here you will be given an option to escape. Hit that left arrow key if you suspect that any important files may match FILE ?. The screen directions say "start with" but only those that match the wild card form FILE ? will be scratched (4532).

If you chose to abort you'll have to scratch the test files FILE 1, FILE 2, FILE 3, and FILE 4 as well as COMBO 2 and HOLD manually (okay, okay—not a scraper just an immediate mode disk command (4500)).

If you chose not to abort by hitting return, you should be seeing in message that the files have been scratched followed by the revised directory. You may see FILE NEW or FILE2 since those don't match the wild card pattern. The program directions were a bit of an overkill.

6328 SCRATCH: COMBO ?



This phase will really only scratch COMBO 2 unless there are other files that match the pattern (4532). COMBO ? HOLD will not be scratched and there should be no COMBO 1 to be scratched.

Again use the "left arrow" key to abort if you are in doubt. Scratch manually before you run again.

6329 The End

You see the directory showing only HOLD if you chose to continue and the program ends. You must scratch HOLD before you can run this program to this point again.

If your wedge is active you can use the "pre-printed" screen message. Simple "cursor up" and hit RETURN.

Can you guess where you'll get into trouble if you run again before scratching HOLD? If not, run it again and find out.

6330 LISTING of "COMMAND DEMO"

Rem: This program is obviously longer than the crude programs in (6200) because it is kinder to the user and has nicer displays.

This listing and the printer produced lines in (6340) were prepared with a different printer whose interface changes the graphic characters for display controls into a more readable form. This makes the listing look longer than it would otherwise.

```

10 REM ***** COMMAND DEMO *****
15 GOTO150
99 INPUT#15, EN, EM$, ET, ES: RETURN
100 REM *** VIEW DIRECTORY ***
101 OPEN1, B, O, "$": P$="#"
102 GET#1, A$, B$
105 GET#1, A$, B$
106 GET#1, A$, B$
110 C=0: IFA$<>" " THEN C=ASC(A$)
112 IFB$<>" " THEN C=C+ASC(B$)*256
114 PRINTC; TAB(5);
116 GET#1, B$: IFST<>0 THEN 140
118 IFB$=CHR$(34) THEN PRINTB$;
119 IFB$<>CHR$(34) THEN 116
120 GET#1, B$: IFB$<>CHR$(34) THEN PRINTB$; : GOTO120
122 IFB$=CHR$(34) THEN PRINTB$;
130 GET#1, B$: IFB$=CHR$(32) THEN PRINTP$; : GOTO130
132 PRINTTAB(32); : C$=""
134 C$=C$+B$: GET #1, B$: IFB$<>" " THEN 134
136 PRINTLEFT$(C$, 3)
138 IFST=0 THEN 105
140 PRINT" BLOCKS FREE ": CLOSE1: RETURN

```

The zeros do not have slashes through them.

○ the number zero

□ the letter "oh"

(Continued)

6330 (Cont'd)

```

150 R$=CHR$(13):K1$="{RVON}{GRY2}ANY KEY WHEN READY{RVOF}{GRY1}":POKE53281,15:PO
KE53280,4:PRINT"{CLR}{GRY1}"
152 K1$=R$+"{RVON}{GRY2} ANY KEY WHEN READY {RVOF}{GRY1}"+R$
155 K2$=R$+"{RVON}{GRY2} {GRY1}RETURN{GRY2} TO CONT / {GRY1}←{GRY2} TO ESCAP
E{RVOF}{GRY1}"+R$
200 PRINT" {RVON}COMMAND DEMO -- DIRECTIONS{RVOF}":FORT=1T0500:NEXT
202 PRINTR$" [1]. {RVON}CHOOSE PRACTICE DISK"R$
204 PRINT" DO {RVON}NOT{RVOF} USE ONE THAT CONTAINS:"R$R$TAB(6)"FILE 1"TAB(20)"
COMBO 1"
205 PRINTTAB(6)"FILE 2"TAB(20)"COMBO 2"R$TAB(6)"FILE 3"TAB(20)"HOLD"
206 PRINTTAB(6)"FILE 4"R$R$
207 PRINT" {RVON}USE{RVOF} A FORMATTED DISK OF COURSE!"R$
208 PRINT" FOR BEST RESULTS:"R$" USE A DISK THAT HAS VERY FEW ENTRIES"R$
209 PRINT" [2] {RVON}INSERT PRACTICE DISK"R$R$K1$
210 GETK$:IFK$=""THEN210
220 OPEN15,8,15,"I":GOSUB99:IFEN<20THEN230
225 PRINTEN;EM$,ET"-ES:CLOSE15:END
230 PRINT"DISK DIRECTORY"R$R$:GOSUB100:PRINTK2$
240 GETK$:IFK$=""THEN240
241 IFK$="←"THENCLOSE15:END
242 IFK$<>R$THEN240

```

```

Rem: 250 WRITE
      270 READ
      300 COMBINE (COPY command)
      350 COMBINE (COPY command)

```

```

250 PRINT"{CLR}TO WRITE TEST FILES"
252 FORI=1T04:F$(I)="FILE"+STR$(I)
254 OPEN6,8,6,"O:"+F$(I)+",S,W":GOSUB99
256 IFEN<20THEN260
258 IFEN=63THENPRINTEM$;:CLOSE6:CLOSE15:PRINT" ON THIS DISK.":END
259 PRINTEN;EM$,ET"-ES:CLOSE6:CLOSE15:END
260 INPUT"DATA(2 ITEMS-KEEP 'EM SHORT)":D1$,D2$
262 PRINT"WRITING "F$(I):PRINT#6,D1$R$D2$R$F$(I)
264 CLOSE6:NEXT
266 PRINT"HERE'S THE CHANGE IN THE DIRECTORY":GOSUB100:PRINTK1$
268 GETK$:IFK$=""THEN268
270 PRINT"{CLR}TO READ TEST FILES"
272 FORI=1T04:F$(I)="FILE"+STR$(I)
274 OPEN6,8,6,"O:"+F$(I)+",S,R":GOSUB99
276 IFEN<20THEN280
278 IFEN=62THENPRINTEM$;:CLOSE6:CLOSE15:PRINT" NOT ON THIS DISK.":END
279 PRINTEN;EM$,ET"-ES:CLOSE6:CLOSE15:END
280 INPUT#6,D1$,D2$,D3$
282 PRINT"READING "F$(I):PRINTD1$R$D2$R$D3$R$
284 CLOSE6:NEXT
286 PRINT"NO CHANGE IN THE DIRECTORY":GOSUB100:PRINTK1$
288 GETK$:IFK$=""THEN288
300 PRINT"{CLR}TO COMBINE THE TEST FILES"
310 PRINT#15,"CO:COMBO 1=0:FILE 2,0:FILE 4"
315 GOSUB99:IFEN<20THEN320
316 PRINTEN;EM$,ET"-ES:CLOSE6:CLOSE15:END
320 PRINTR$R$"TO READ COMBO FILE"
325 OPEN6,8,6,"O:COMBO 1,S,R":GOSUB99
326 IFEN<20THEN330
327 PRINTEN;EM$,ET"-ES:CLOSE6:CLOSE15:END
330 INPUT#6,D$:PRINTD$:IFST=0THEN330
335 PRINT"READING COMPLETE":CLOSE6
340 PRINT"HERE'S THE CHANGE IN THE DIRECTORY":GOSUB100:PRINTK1$
345 GETK$:IFK$=""THEN345
350 PRINT"{CLR}TO COMBINE THE TEST FILES ANOTHER WAY"

```

(Continued)

6330 (Cont'd)

```

360 PRINT#15,"CO:COMBO 2=0:FILE 1,0:FILE 2,0:FILE 3"
365 GOSUB99:IFEN<20THEN370
366 PRINTEN;EM$,ET"-ES:CLOSE6:CLOSE15:END
370 PRINTR$R$"TO READ COMBO 2 FILE"
375 OPEN6,8,6,"O:COMBO 2,S,R":GOSUB99
376 IFEN<20THEN380
377 PRINTEN;EM$,ET"-ES:CLOSE6:CLOSE15:END
380 INPUT#6,D$:PRINTD$:IFST=0THEN380
385 PRINT"READING COMPLETE":CLOSE6
390 PRINT"HERE'S THE CHANGE IN THE DIRECTORY":GOSUB100:PRINTK1$
395 GETK$:IFK$=""THEN395

```

```

Rem: 400 Rename
      450 Scratch
      470 Scratch

```

```

400 PRINT"(CLR)TO RENAME A FILES"
410 PRINT#15,"RO:HOLD=COMBO 1"
415 GOSUB99:IFEN<20THEN420
416 PRINTEN;EM$,ET"-ES:CLOSE6:CLOSE15:END
420 PRINTR$R$"TO READ HOLD FILE"
425 OPEN6,8,6,"O:HOLD,S,R":GOSUB99
426 IFEN<20THEN430
427 PRINTEN;EM$,ET"-ES:CLOSE6:CLOSE15:END
430 INPUT#6,D$:PRINTD$:IFST=0THEN430
435 PRINT"READING COMPLETE":CLOSE6
440 PRINT"HERE'S THE CHANGE IN THE DIRECTORY":GOSUB100:PRINTK1$
445 GETK$:IFK$=""THEN445
450 PRINT"(CLR)TO SCRATCH THE TEST FILES"
451 PRINTR$R$(RED){RVON}      W A R N I N G ! !      (GRY1)"R$R$"
452 PRINT"THIS PHASE WILL SCRATCH ALL FILES AND"R$"PROGRAMS WHOSE NAMES START";
453 PRINT" WITH {RVON}FILE "R$K2$
455 GETK$:IFK$=""THEN455
456 IFK$=""←"THENCLOSE6:CLOSE15:END
457 IFK$<>R$THEN455
460 PRINT#15,"SO:FILE ?"
461 GOSUB99:IFEN<20THEN464
462 PRINTEN;EM$,ET"-ES:CLOSE6:CLOSE15:END
464 PRINTET;EM$R$K1$
465 GETK$:IFK$=""THEN465
466 GOSUB100:PRINTK1$
468 GETK$:IFK$=""THEN468
470 PRINT"(CLR)TO SCRATCH THE COMBO FILES"
471 PRINTR$R$(RED){RVON}      W A R N I N G ! !      (GRY1)"R$R$"
472 PRINT"THIS PHASE WILL SCRATCH ALL FILES AND"R$"PROGRAMS WHOSE NAMES START";
473 PRINT" WITH {RVON}COMBO "R$K2$
475 GETK$:IFK$=""THEN475
476 IFK$=""←"THENCLOSE6:CLOSE15:END
477 IFK$<>R$THEN475
480 PRINT#15,"SO:COMBO ?"
485 GOSUB99:IFEN<20THEN487
486 PRINTEN;EM$,ET"-ES:CLOSE6:CLOSE15:END
487 PRINTET;EM$R$K1$
488 GETK$:IFK$=""THEN488
489 GOSUB100:PRINTR$"END OF {RVON}COMMAND DEMO"R$R$"DON'T FORGET TO SCRATCH {RVD
N}HOLD{RVOF}."
490 PRINT"USE COMMAND BELOW IF <WEDGE> ACTIVE"R$R$"@S:HOLD"

```

6340 **Explanation of COMMAND DEMO**

Rem: With few exceptions, explanations will be limited to those lines that affect files or the drive. If your programming skills are very limited you may wish to tackle RELATIVE FILES (7000) first. Come back here later.

Rem: Unlike other programs in this text, the subroutines have been placed at the beginning of the program. Although such is not the case here, in very long programs, time can be saved with this technique.

Rem: You'll notice that this program has been "crunched" a great deal more than most of the other file programs. Hope that the explanation isn't too hard to follow.

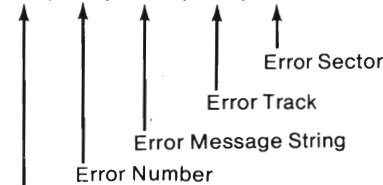
Rem: If you are a beginner, don't worry if you don't understand everything the first time around. You can go on to RELATIVE FILE (7000).

Rem: As usual references to the text are enclosed in "()" but those to lines in this program are not.

Rem: Commands from (4000) are in lines 220, 310, 360*, 410, 460, 480*.

6341 **INITIAL SET UP**

```
10 REM ***** COMMAND DEMO *****
15 GOTO150
99 INPUT#15, EN, EM$, ET, ES: RETURN
```

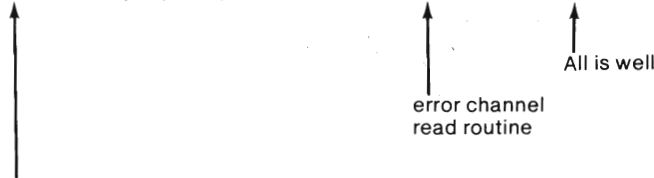


The file will be opened before we call this subroutine.

Rem: The Subroutine, lines 100 - 140, is explained in (6347) with extension in 6400.

150-210 are simple for screen display. (6330). Simply keep in mind that R\$=CHR\$(13) is the carriage return. R\$ is used throughout to control the display and for file data format. (5430), (6263)

```
220 OPEN15, 8, 15, "I": GOSUB99: IF EN < 20 THEN 230
```



Opens the command channel and INITIALIZES (4300) the disk. The user's data disk could get turned into a real mess if its ID code happened to match that of the disk from which the program was loaded. (Whew! They tell me to never end a sentence with a preposition).

```
225 PRINTEN, EM$, ET "-" ES: CLOSE15: END
```

The error channel read routine will pick up errors. Those commonly expected are 74 Drive Not Ready (user forgot to insert the disk or close the door, etc.) and possible some in the 20's (user forgot to format disk). Other causes are possible of course.²

* - Not included in this section since they are repetitious. See main listing (6320).

² - Hardware failure, damaged disk, etc. Use 9500 and/or your manual.

6341 (Cont'd)

```
230 PRINT"DISK DIRECTORY"R*R*:GOSUB100:PRINTK2*
240 GETK$:IFK$=""THEN240
241 IFK$="←"THENCLOSE15:END
242 IFK$<>R$THEN240
```



240 - 242 is a more sophisticated "wait for user" routine. We could call it a mini menu. Notice 241 closes the command channel if the user chooses to abort.

6342 FILE WRITE

```
250 PRINT"{CLR}TO WRITE TEST FILES"
252 FORI=1TO4:F$(I)="FILE"+STR$(I)
254 OPEN6,8,6,"O:"+F$(I)+",S,W":GOSUB99
```



```
256 IFEN<20THEN260 ← all clear. Line 260 starts data input.
```

```
258 IFEN=63THENPRINTEM*::CLOSE6:CLOSE15:PRINT" ON THIS DISK.":END
```

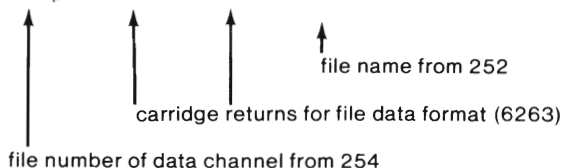


Here we give the user a special message. In other kinds of programs we might branch to the menu or some special place in the program.

```
259 PRINTEN;EM*,ET"-ES:CLOSE6:CLOSE15:END
```

This is the standard error message. Like line 258 we need not end the program. The main purpose of lines 258 and 259 is to show you that such things can be done.

```
260 INPUT"DATA(2 ITEMS-KEEP 'EM SHORT)":D1$,D2$
262 PRINT"WRITING "F$(I):PRINT#6,D1*R$D2*R$F$(I)
```



```
264 CLOSE6:NEXT
```

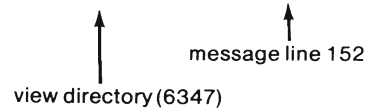
We must CLOSE the file before we repeat the loop. (6143). If you like experiments, switch. i.e. Use NEXT:CLOSE6. Run the program: Use the SCRATCH command (4500) to clean up the mess. Remember that * in the directory denotes a file that has been left open. (6253)

6342 (Cont'd)

```

266 PRINT"HERE'S THE CHANGE IN THE DIRECTORY":GOSUB100:PRINTK1$
268 GETK$:IFK$=""THEN268

```



Compare this routine to that in 230 - 242

6343 FILE READ

Lines 270 - 288 parallel one another. Rather than repeat, I suggest you turn to those lines in the main listing, (6330) and compare them line-by-line.
 254 & 274, 258 & 278, 260 & 280, 262 & 282

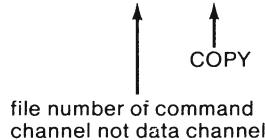
6344 COMBINE FILES Copy to append (4800)

Rem: COMMAND DEMO finally starts to demonstrate commands.*

```

310 PRINT#15,"CO:COMBO 1=0:FILE 2,0:FILE 4"

```



The COPY command operates much like string addition. The format is just a bit messier. The manual says we can combine up to four files. When I tried that with these file names, I got 32 SYNTAX ERROR, (command too long). Try it with shorter file names when you get a chance.

```

330 INPUT#6,D$:PRINTD$:IFST=0THEN330 Part of the READ COMBO 1.

```



This line shows a more elegant way of finding the end of a file than the flag system used in (6200).

ST = 0 means there is more in the file.

ST = 64 signals the end of the file.

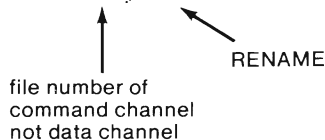
350 - 395 are a repetition of 300 - 345.

6345 RENAME (4600)

```

410 PRINT#15,"RO:HOLD=COMBO 1"

```



I like this command. I used it to rename my «wedge» with a single letter name. I'm lazy.

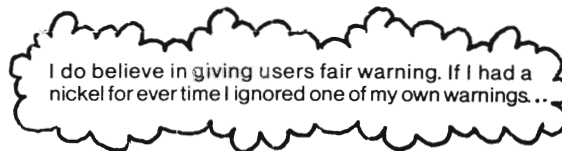
* - Lines 300, 315 - 327, 335 - 345 are basic BASIC or similar lines have already been described.

6346 SCRATCH (4500)

```

450 PRINT"(CLR)TD SCRATCH THE TEST FILES"
451 PRINTR$R$(RED)(RVON)  W A R N I N G  !!  (GRY1)"R$R$
452 PRINT"THIS PHASE WILL SCRATCH ALL FILES AND*R$*PROGRAMS WHOSE NAMES START";

```



```
460 PRINT#15,"SO:FILE ?"
```

↑
file number of
command channel
not data channel

↑
wild card symbol (4532)

Lines 470 - 488 parallel 450 - 468 but SCRATCH the files that match COMBO? instead. 486 shows how we can use the output from the error read routine as a "user" message.

The End - we are stuck with HOLD in the directory. 489 - 490 are user convenience lines.

6347 Subroutine*

```
100 REM *** VIEW DIRECTORY *** ← Subroutine
```

Rem: This subroutine was adapted from "DIR" on the "TEST/DEMO" disk. If you are still a beginner programmer, don't worry if there are parts in here that you don't quite "get." Keep thinking, it will come. If you are a bit more advanced, the mechanics should not present any major problems. You'll need a careful study of (0270) with special attention to directory charts. The following section (6400) addresses some of the contradictions you may spot as you go.

```
101 OPEN1,8,0,"$":F$="#"
```

↑
fillers that you saw in the directory display.

notice that the channel is one of those usually reserved for DOS (5312). Recall that the directory is not really a data file and that it can be loaded like a program (1201).

```

102 GET#1,A$,B$  These lines skip past information in the directory entry or header
105 GET#1,A$,B$  that we don't need to see in normal operations.
106 GET#1,A$,B$

```

```

110 C=0: IFA$<>" " THEN C=ASC(A$)  Here it computes the number of blocks used by the
112 IFB$<>" " THEN C=C+ASC(B$)*256  file. (7335) contains more information on lo and hi
114 PRINTC;TAB(5);  bytes.

```

```
116 GET#1,B$: IFST<>0 THEN 140  140 is the end of the subroutine.
```

The STATUS function is used as in line 330 (6344) to see if the job is done.

(Continued)

6347 (Cont'd)

```
118 IFB$=CHR$(34) THENPRINTB$;    It's looking for the 1st quote mark in the file name.
119 IFB$<>CHR$(34) THEN116
```

CHR\$(34) is a quote mark. i.e. 34 is the ASCII code for ".

```
120 GET#1, B$: IFB$<>CHR$(34) THENPRINTB$; :GOTO120
```

If it finds something other than a quote mark, it prints it and repeats. This is the file name in the directory.

```
122 IFB$=CHR$(34) THENPRINTB$;    It finds the last quote mark.
```

```
130 GET#1, B$: IFB$=CHR$(32) THENPRINTP$; :GOTO130
```

↑
the marker from line 101

Space is always reserved for file name. (32 is ASCII for a space).

```
132 PRINTTAB(32); :C$=""
134 C$=C$+B$: GET #1, B$: IFB$<>" " THEN134
136 PRINTLEFT$(C$, 3)
```

Here it's getting the file type (SEQ, PRG etc) or ID code if still in header.

```
138 IFST=0 THEN105
```

If there is more in the file, it goes to 105 to skip 2 bytes and continue getting characters.

```
140 PRINT" BLOCKS FREE ":CLOSE1:RETURN
```

↑
Or the program will stop at the next request

6348 The Truth

Rem: This program has bothered me for some time. I promised I'd tell you when I wasn't completely sure of something and this is one of those times.

The routine itself is too useful to omit because I'm not 100% sure of why it works. We know it works and that's enough for most people. If you don't want to open a can of worms, skip the rest of this chapter.

6349 The Problems (0260 and 0270)

- 1) The most obvious problem if you've studied (0273) is where are the 3 bytes that are used for file type? They are on my screen but not in (0273). There is the one byte at the beginning that denotes file type and indicates a closed file. This program is not "fancy" enough² to do anything with that. I'm wondering if they are hiding in the "unused" part.
- 2) The "spacers," lines 102 - 106, are the real key to the whole thing. Notice that on the 1st pass 4 bytes are skipped and on subsequent passes (hitting line 138) only 2 are skipped. These bytes appear to be the 2 bytes between directory entries (0272) that we would encounter during all but the 1st pass. There are other pairs of bytes that we don't see in the 30 byte directory entry.

How the first pass works with the directory header is what really has me going. The directory header is supposed to start on Track 18, sector 0, byte 144. If it does, and lines 102 - 105 skip 4 bytes the program must start earlier than byte 144. BAM (0271) is supposed to occupy those bytes and we do have a problem.

The questions...

- 1) Where on Track 18, Sector 0 does this program start or does the header really start at byte 144?
- 2) Does using the DOS channel 0 have anything to do with it?
- 3) Are the file type characters in the 30 bytes for directory entry after all?

6400 "A can of worms"

Rem: This section is for those whose curiosity got really piqued by (6349). Most users will want to put this section on a back burner. If you're not real excited about this go on to RELATIVE FILES.

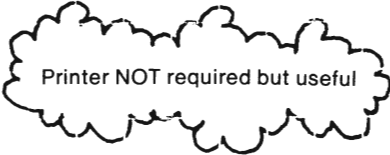
Rem: If you want simple answers to the questions in (6349) I fear you'll be disappointed. The program only gives you the tools for exploration. Only one of the questions will be answered and more questions will be raised.

6410 Program Objectives

Present a more thorough view of the directory.

Provide hard copy of same.

6420 User Instructions:



Printer NOT required but useful

6421 Hardware requirements: (optional)

In addition to your usual system a 1525 printer or other printer with a suitable interface would be nice but is NOT required.

6422 Name: EXP\$-PRINT

load type: (normal)

² - It's not playing any binary AND/OR game with bits within the byte.

6423 Load the program.

6424 If you do not want a screen dump of the results, remove the "REM" from line 999.

Warning. The screen dump routine² is in BASIC and is very slow. You may want to try the program without the printout a few times. If you have a better screen dump routine, you may wish to replace this one.

If you do not remove the "REM" from 999 and you do not have a printer, it's no big deal. Your program will simply end with a "device not present" message.

If you do have a printer, pull a listing of the directory part of the program. It will be easier to study that way.

double spaced		single spaced
OPEN200, 4: CMD200	or	OPEN4, 4: CMD4
LIST-150		LIST-150
PRINT#200: CLOSE200		PRINT#4: CLOSE4

6430 LISTING of EXP\$-PRINT

Rem: The line numbers correspond to those used in COMMAND DEMO. Many have been changed but basically perform the same task. A few have been added.

```

100 n$="0123456789 123456789 123456789 123456789"
101 print"(CLR)"n$:open1,8,0,"$":x$="@":y$="*":z$="#"
102 get #1,a$,b$:print"12"asc(a$+chr$(0))asc(b$+chr$(0))
105 get #1,a$,b$:print"34"asc(a$+chr$(0))asc(b$+chr$(0))
106 get #1,a$,b$
107 ifa$=""thenprint"a";
108 ifb$=""thenprint"b";
110 c=0:ifa$<>""thenc=asc(a$)
111 ifa$<>""thenprintasc(a$)
112 ifb$<>""thenc=c+asc(b$)*256
113 ifb$<>""thenprintasc(b$)
114 printmid$(str$(c),2)x$;
116 get#1,b$:ifst<>0then140
117 ifb$=chr$(32)thenprint".";
118 ifb$=chr$(34)thenprint"[";
119 ifb$<>chr$(34)then116
120 get #1,b$:ifb$<>chr$(34)thenprintb$;:goto120
122 ifb$=chr$(34)thenprint"]";
130 get #1,b$:ifb$=chr$(32)thenprinty$;:goto130
132 c$=""
134 c$=c$+b$:get #1,b$:ifb$<>""then134
136 printleft$(c$,3)z$
138 ifst=0then105
140 print" blocks free":close1
150 rem end

```

(Continued)

² - Adapted from the 1525 User's Manual. Corrections have been made.

6430 (Cont'd)

```

999 rem end
1000 rem ***** screen dump2 *****
1010 si$=chr$(15):b$=chr$(8):po$=chr$(16)
1020 rv$=chr$(18):ro$=chr$(146):qt$=chr$(34)
1030 mf$=chr$(145):vr=peek(648)*256
1040 open4,4:print#4
1050 forcl=0to24:qf=0:as$=mr$:forro=0to39
1060 sc=peek(vr+40*cl+ro)
1070 ifsc=34thenqf=1-qf
1080 ifsc<>162then1110
1090 qf=1-qf:ifqf=1thenas$=as$+rv$+qt$:goto1170
1110 ifqf=1and(sc>=128)thensc=sc-128:goto1130
1120 ifsc>=128thensc=sc-128:rf=1:as$=as$+rv$
1130 ifsc<32orsc>95thenas=sc+64:goto1160
1140 ifsc>31andsc<64thenas=sc:goto1160
1150 ifsc>63andsc<96thenas=sc+32:goto1160
1160 as$=as$+chr$(as)
1170 ifrf=1thenas$=as$+ro$:rf=0
1180 nextro
1190 ifqf=0thenprint#4,si$po$"20"as$:goto1210
1200 print#4,si$+po$+"20"+as$+as$+qt$
1210 nextcl:print#4,si$:close4

```

6440 **SAMPLE RUN**

Rem: 3-digit numbers refer to lines in EXP\$-PRINT rather than COMMAND DEMO

0123456789 123456789 123456789 123456789

column counter

line 102 → 12 1 4

line 105 → 34 1 1

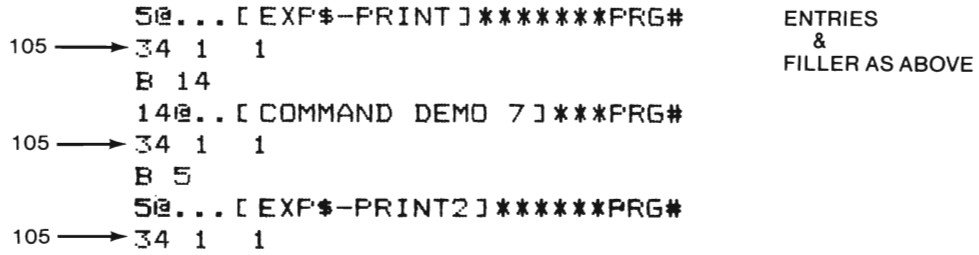
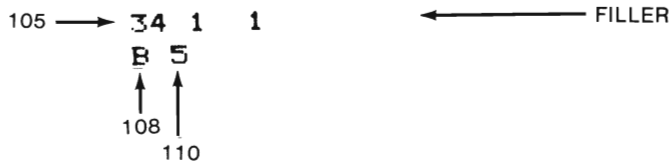
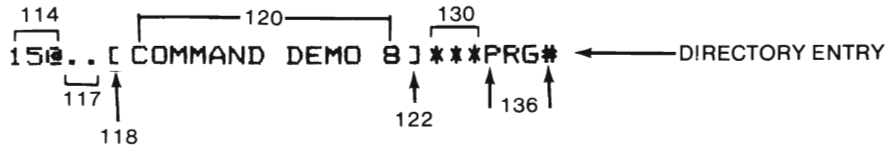
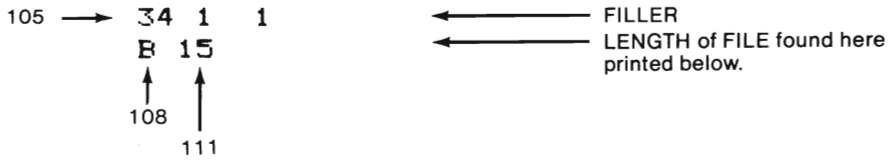
← ????????????

← DIRECTORY HEADER

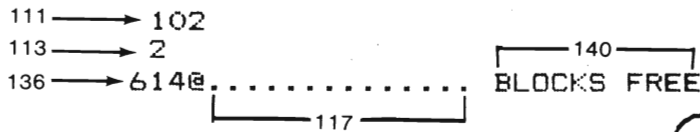
(Continued)

² - Adapted from the 1525 USER'S MANUAL. Corrections have been made.

6440 (Cont'd)



108 not applicable



614 = 102 + 2*256

6450 **AFTERTHOUGHTS**

The questions in (6349)...

6451 1) If the header's first quote is at byte 144, the program doesn't start there. The program starts 6 bytes in front of that quote mark. Lines 107 and 108 show two empty bytes in front of the quote mark. Sorry that I don't have the knowledge to answer.

6452 2) This program does not work with other channels. Try it. Also try it with the name of a program instead of \$ in line 101.

6453 3) Those characters (PRG, SEQ, REL) come through loud and clear just before we skip the two bytes in 105. I played with the program a bit. Look at the results in 6460.

6454 A new question...

Are the bytes for the length of the file really at the end as shown in (0273)? I thought I had that resolved but a few more tests will be necessary.

6455 A surprise...

The DOS keeps track of the blocks free where we would expect to find the length of the next file. I knew it had to be someplace but was never sure which.

6456 Something I should have known...

The value of C from 110 - 114 accounts for the 0 we always see in front of the header. This 0 conveniently prevents lock ups when we try to "run the directory."

6457 Another new question...

Why do there always seem to be 14 passes through 117? Just before the directory file ends?

6458 What I'd like to do now...

1) Use random files and ASCII translations to do a thorough examination of Track 18, Sector 0 and Sector 1. I fear I'd need a computer to analyze the results.

2) Get ahold of the original author of this program and the 1541 DOS designer and get some reliable information. I'm seriously bothered by possible inaccuracies in all of (0270). At least it gives us an idea of how things may work.

3) Count the characters it "GETS" but does not do anything with during each part of the program.

6459 Notice: An effective, elegant and efficient program is not always easy-to-follow.

6460 "That's all she wrote"

6461 The RUN

0123456789 123456789 123456789 123456789

```

12 1 4 34 1 1 ABO@[ BACKUP2          ]*←
  78 N2 #34 1 1 B 22
22@...[ ON LOADING ]*****← 80 PRG#34 1 1
  B 49
49@...[ GB ]*****← 80 PRG#34 1 1
  B 11
11@...[ N114 ]*****← 82 REL#34 1 1
  B 1
1@...[ SN114 ]*****← 83 SEQ#34 1 1
  B 4
4@...[ 114PAGE AFRP ]*****← 80 PRG#34 1 1
  B 1
1@...[ 0114 ]*****← 83 SEQ#34 1 1
  B 1
1@...[ 1114 ]*****← 83 SEQ#34 1 1
  B 1
RUN1000
    
```

The numbers following the left arrows are due to 130 below. 131 doesn't seem to happen here. Do you see what I see?
 80 PRG
 82 REL
 83 SEQ
 Try with several disks. The 78 after the header is not consistent like the 80, 82 and 83. File type?????

6462 The Changes to EXP-PRINT

```
116 GET#1,B$:IFB$<>OTHENPRINT"116  ":GOTO140
```

just a flag to make sure 116 ends the program i.e. sends to 140.

```

130 GET #1,B$:IFB$<>"ANDB$<>CHR$(32) THENPRINT"←"ASC(B$);
131 IFB$="" THENPRINT" ";
132 IFB$=CHR$(32) THENPRINT"Y$";:GOTO130
133 C$=""
    
```

Account for the changes in the display.

6463 P.S. See if removing LEFT\$ from 136 picks up the DOS and format type. (The 2A we usually see in the directory header.

6499 Rem: I played with **EXP\$-PRINT** a bit and got these results. I'm sorry to leave you with a can of worms but if I wait until all conflicts are resolved this book will come out when you are in fact a computer. That's really the joy of these silly things. The more you know, the more there is to know.



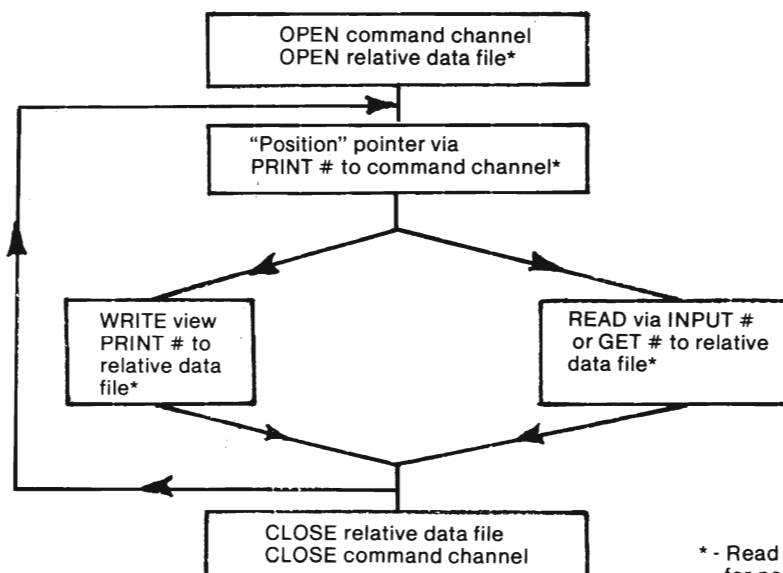
7000 RELATIVE FILES**CHAPTER DIRECTORY**

7100	REL FILES in GENERAL
7110	Overview
7120	Anatomy/Vocabulary
7130	Records
7140	Command Format and Function
7150	Special Error Messages
7200	"GEN REL" - Sample Program
7210	Program Objectives
7220	Job Descriptions
7230	User Instructions - Preliminary
7240	User Instructions - First Run
7250	User Instructions - General
7260	Record Structure
7270	User Adaptations
7280	Unwanted Relative Files
7300	"GEN REL" - Detailed Explanation
7310	Initial Set Up
7320	Jobs
7330	Subroutines
7340	Secondary Program
7400	"GEN REL" - Complete Listing

7100 Relative Files in General

Rem: Relative Files allow user access to individual records and even to fields within those records. The rest of this section is devoted to qualities regarding relative files. You may find it necessary to refer to DATA FILES in GENERAL (5000) especially (5120), (5200) and (5300).

Feel free to postpone this section until after 7200. You can make use of the sample program even if you have no interest in programming.

7110 Procedure Overview

* - Read command channel for normal and unscheduled error conditions.

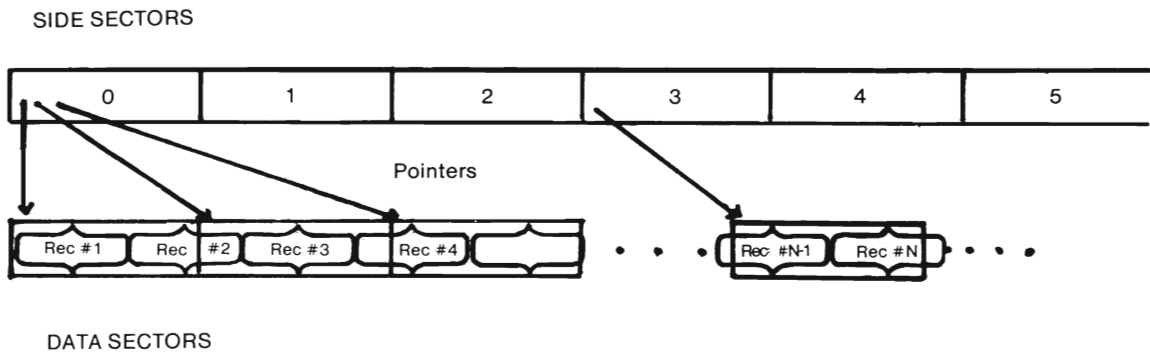
7111 Notice - unlike sequential files (6110) we need not close and reopen when we change from read to write or vice versa.

7112 Notice - we do have to "position" a pointer prior to each operation (unless we want to work in sequence). The open to command is REQUIRED for relative file operation and not just to check for errors. (7143)

7113 Notice the frequent reading of the command (error) channel. At times we'll be looking for special messages. (7150)

7120 **ANATOMY & VOCABULARY**

Relative files use two kinds of sectors.



7121 DATA SECTORS are structured in the usual manner. ie. 2 bytes for track and sector of next data sector and 254 bytes for data. (235)

7122 SIDE SECTORS are peculiar to relative files for DOS bookkeeping. Each of the six side sectors contains:

- 2 bytes Track and sector of next side sector
- 1 byte its own number
- 1 byte record length
- 12 bytes location table (track and sector of all six side sectors)
- 240 bytes pointers to 120 data blocks (track and sector)

6 side sectors * 120 datablocks = 720 data blocks.
 sidesector



664 blocks free - 6 side sectors = 658 data blocks

Theoretically we can access more blocks than the disk contains. 5 side sectors would allow access to only (5*120) = 600 blocks, hence the extra.

658 blocks for data * 254 data byte = 167,132 bytes for data.
 block

7130 **RECORDS**

Rem: A record is simply a batch of data, but the concept of an individual record is critical to relative files. Although the following are explored in detail in later sections, a few generalities are in order now.

7131 **RECORD LENGTH (254 byte maximum)**

All records in a particular relative file are of equal length. When the file is created, this number is placed in the directory entry. The record length as set by the programmer is also contained in each side sector used.

When the record length does not divide 254 evenly, records will "span" sectors. Fortunately the DOS takes care of the bookkeeping so this overlapping from one sector to the next is not a programmer problem.

7132 **RECORD NUMBER (??max)**

The programmer usually assigns a number to each record. To get at a particular record, the number and a simple algorithm are used to "position" the pointer. (7335)

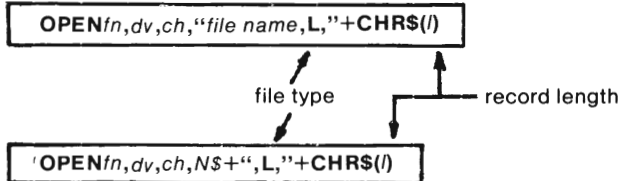
A balance between record length and data blocks available determine the maximum number of records possible in a relative file. (7273)

7133 **FIELDS** are subdivisions of records. Field lengths may vary within records from a single byte to the length of the record. The programmer must consider these variations when positioning for read or write operations. Individual fields can be accessed for both reading and writing. There are some cautions on writing fields (7322). Field structure may vary from record to record so long as the programmer accounts for differences in read and write operations. The programmer should consider a balance between adequate room for user input and too much dead space. In relative files, unused bytes are padded since record length remains constant throughout the file. When setting field lengths, the programmer must allow room for "separations" (carriage return or comma, (5400). See also Record Structure (7273).7140 **Command Format & Function**

Rem: This section is intended as a general reference. The best road to mastery is the sample program, (7200), then the detailed explanation (7300)

7141 **OPEN to Create and/or Use**

format:



parameters: *fn* = file number, *dv* = device number, *ch* = data channel number as described in "FILES" (5310)

"file name" or *N\$* is the name of the file.

","L," indicates relative file type. (compare to "S," in 6120 & 6130)

l = length of each record in the file.

function: if no file of the name specified exists on the disk, this command creates the directory entry and opens the usual communication link. With relative files the length of each record and the location of the 1st side sector are included in addition to the usual information. (273)

If a relative file of the name and length specified exists, this command simply opens the communication link and can be replaced by the short form below.

Disagreement of file type or record length results in an error condition.

7142 **OPEN to Use**

format: `OPENfn,dv,ch,"file name"` or `OPENfn,dv,ch,N$`

parameters: as on previous page (7141)

function: If a relative file by the name specified exists on the disk, this command opens the communication link. The DOS checks file type and reads record length from the directory entry. This command can be used in a file examine routine if the name but not record length is known. (7340)

7143 **"Position" to any record**

Rem: Assume the command, `OPEN 15,8,15` has been sent to open the command channel.

format: `PRINT# 15,"P"CHR$(ch)CHR$(lo)CHR$(hi)`

parameters: 15 = file number in the OPEN command channel statement.

"P" = command message string (5330)

ch = channel number of the relative data file

lo = low byte of record number (7335)

hi = high byte of record number (7335)



Function: This command "positions" a pointer to the beginning of the record determined by **lo** and **hi**². It, or the command below, is sent before read or write operations. If not used, the file will be accessed sequentially, starting with Record # 1.

7144 **"Position" to any byte within record.**

format: `PRINT# 15,"P"CHR$(ch)CHR$(lo)CHR$(hi)CHR$(p)`

parameters: *p* = byte number as counted from the beginning of the record (7260).

Others as above.

function: This command positions the pointer to the byte (*p*) of the record number specified (*lo* & *hi*). If *p* = 1 this command is equivalent to the command above (7133).

² These parameters are described in detail in (7335)

7145 Other Commands.

```
PRINT#fn, ---
INPUT#fn, ---
GET#fn, ---
```

After "positioning," these read/write commands are used in the normal manner (5330), (5340), (5400) and sample programs (7300 and 6220)

fn = file number of relative file

```
CLOSE#fn
```

This command is critical (as usual). Unlike sequential files we need not close and reopen to change operations (5320)

7150 Special Errors Messages

Rem: These messages can be used as programmer/user protectors rather than "foul up" indicators. (Consider a user spending hours typing data into a file only to find that there ain't enough room in the disk for the last 2 records).

7151 File Creation:

After the "Open to Create" command, it is normal practice to position the pointer to the last anticipated record in the file. The DOS will then mark off room on the disk. It marks the first byte of the record and pads the remaining bytes. If the DOS cannot find room on the disk it will send error #52 - "file too large". The user will know that either the number of records must be reduced or a disk change is required. Records so marked "exist" but are not written. Records marked by the position command can be written. Use of INPUT# can cause a "string too long" BASIC error condition.

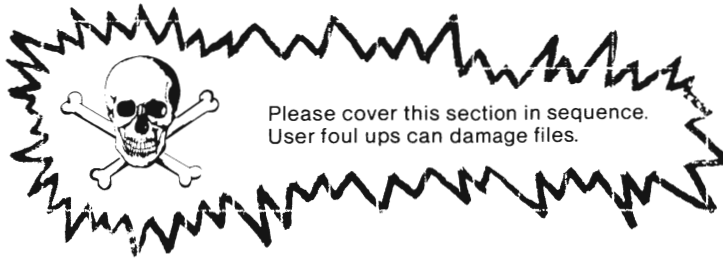
7152 File Expansion:

If the pointer is "positioned" past the end of the file, error # 50 - "record not present" pops up to say the record does not exist. It's fair to write this record. If this is done all intermediate records are marked as described above. This "expands" the file, of course making sure that there is room for it. Neither INPUT# or GET# can be used. Error # 50 will just come right back. Print # is allowed. The sample program shows how to extend the "error channel read" routines for use with relative files. (7338)

7153 The DOS will send error # 51 - "Overflow in record" if the programmer allows the user to write too much information into a record in one PRINT# statement². The sample program shows how to protect the user and prevent the problem. (7332).

² - It does not seem to protect us from writing too much when we use more than one PRINT# statement. Guess it can't do everything!

7200 SAMPLE PROGRAM - "GEN REL"



Rem: "GEN REL" (GENeral RELative file program) is intended to provide experience in using a file management program, an easily adaptable program, and a learning experience in writing relative file programs.

7210 PROGRAM OBJECTIVES

Rem: "GEN REL" is divided into two parts. The user friendly main program is for user file tasks and the secondary program is for the user in trouble and the explorer.

The main program enables the user to

- 1) Create new files, expand existing files and change from file to file during operation
- 2) Read and write records individually and in sequence
- 3) Edit records and individual fields within those records
- 4) Easily adapt the program to individual needs

The main program protects the user from reading unwritten and/or non-existent records and from inadvertently overwriting existing records. It also provides numerous job escape options.

The secondary program allows the user to examine parts of the file character by character without loading over the main program.

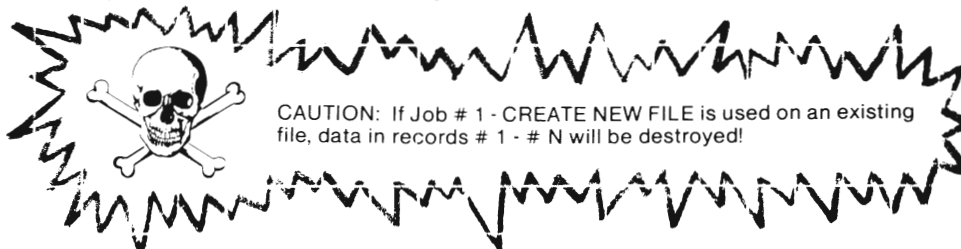
7220 JOB DESCRIPTIONS

7221 CREATE NEW FILE

When selected, the system requests the number of records anticipated (N). The user should allow a few extra and not this number for later reference.

The system will place a marker in the first position of each record from record #1 to record #N as selected. Records so marked are referred to as "existing" but "not written" (does not contain data).

This job must be done before the program will allow read or write operations.

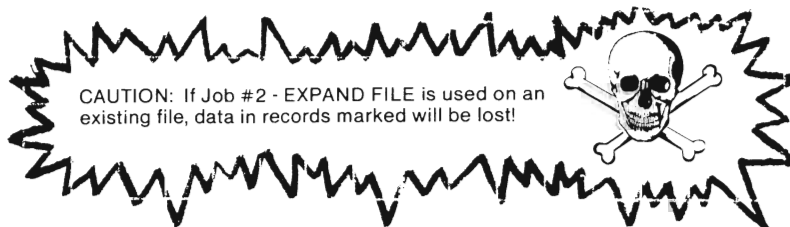


7222 EXPAND FILE

When selected, the system requests the current last record #, S and the number of records to be added, A.

The system will mark off the records from # S + 1 to # S + 1 + A

ex. If the last record was S = 50 and the user inputs A = 8 to be added, the system marks 51, 52, ..., 58.

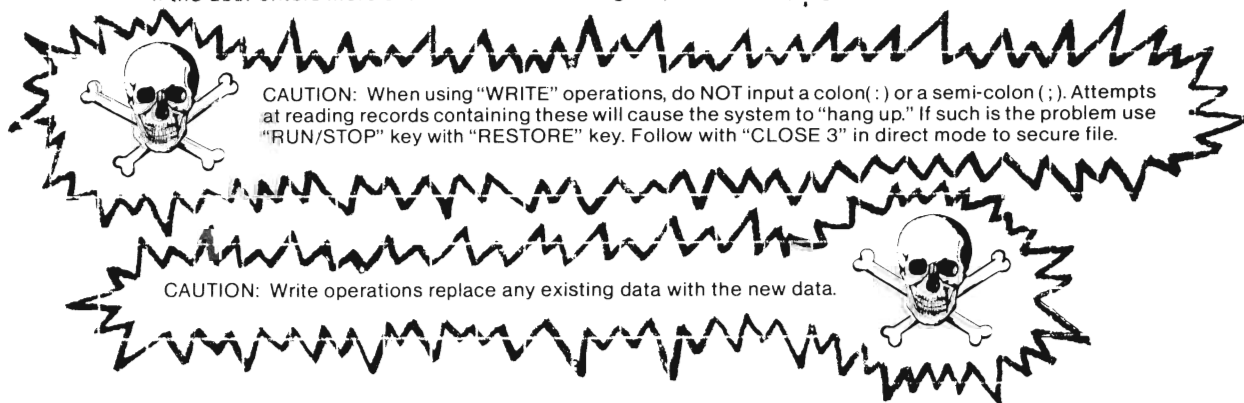


7223 WRITE IN SEQUENCE

When this job is selected, the system request a starting and an ending record number. The records are then presented for writing along with the data category readins and the allowed input length. The process continues until the system finishes or encounters an unwritten (# 'not written') or a not existent record, (# 'not present'). If either of the latter is encountered, the user is automatically returned to the menu. A warning and an escape option are preented when the system finds a record that already contains data. (# 'has been written'). (See below for individual record writing techniques).

7224 WRITE INDIVIDUAL RECORD

When this job is selected, the system requests a record number. After input, the escape option and a warning if the record already contains data is presented. If the user elects to continue, the record is presented for writing. Data category headings and maximum input lengths are shown on the screen, one at a time. The user may enter any characters EXCEPT a colon (:) or a (;). Either of these will cause problems in "send" operations. If the user enters more characters than the length specified the program will truncate.



7225 READ IN SEQUENCE

This job is similar to job # 3 in all matters except that instead of presenting records for writing, the contents are printed to the screen. The system waits for the user before the next record is presented. It returns to the menu when finished, directed by the user, or when a nonexistant and/or an unwritten record is encountered.

7226 READ EXISTING RECORD

This job allows the user to attempt to read any record. If the record has been written it is displayed on the screen. If not, the user is returned to the menu. The 'not present' message means that the record has not been marked by job # 1 or job # 2. i.e. The record does not exist. The 'not written' message means the record has been marked but that it contains no data and is safe for writing.

7227 EDIT EXISTING RECORD

This job allows the user to change an entire record (equivalent to job #4) or change individual fields in the record. When selected, the system asks for a record number, prints the contents of that record to the screen, and presents an "edit menu." The system will return to "the" menu if the record has not been written. The edit menu includes a "change none," a "change all" and a "change field" set of options. Revised contents will be printed until the user selects "change none" to return to the main menu.

7228 CHANGE FILE

When selected, the system asks for a file name and the program begins with the menu. (see "program flow" (7232). The user may then operate on that file.

7229 END RUN

Closes relative data file and command channel and the computer says "READY."

7230 **USER INSTRUCTIONS - Preliminary**

Rem: Whenever you are dealing with files it is a good idea to activate your "WEDGE." (1230). Be sure to work through "First Run" (7240) before doing anything else with the program.

7231 PRELIMINARY DATA

Program Name: GEN REL

Load Type: Normal

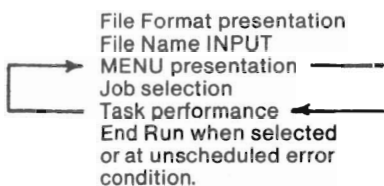
Special Running Instructions: Main Program - RUN

Secondary Program - RUN20000

7232 PROGRAM FLOW

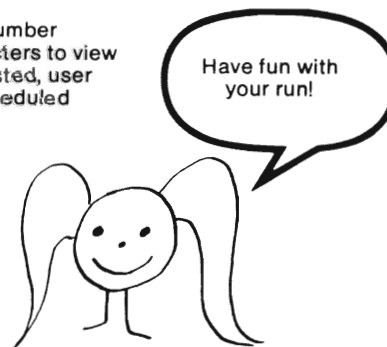
Rem: This is a general description of what happens after you type RUN. You get to do it on the next page so be patient just a little while longer.

Main program:



Secondary program:

INPUT - file name
 starting record number
 number of characters to view
 End run when file exhausted, user request, or unscheduled error condition.



7240 USER INSTRUCTIONS - First Run

Rem: This section assumes that the sample file "GR SAMPLE" has not been altered. i.e., No previous user has performed any operation except reads, change file, or end run. If things do not go as described, you may have to start with "secondary program." (7249)



7241 TO LOAD and RUN PROGRAM

Type: /GEN REL or LOAD"GEN REL",8 if you cannot activate the wedge.

Type: RUN

7242 Now you see FILE FORMAT². Data categories, lengths and starting positions are shown in a table. Don't worry about these now³. All we're going to do here is read the file.

Type: C

² - File Format is explained in record structure (7260) and is used in user adaptation (7270).

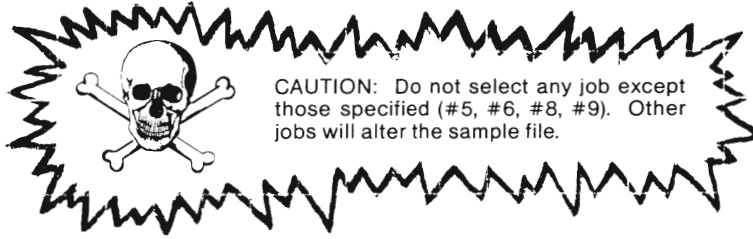
³ - Other processes will be described later (7250).

7243 The system is asking for a file name. Type carefully and check before hitting RETURN. A typing error here will cause an error condition if you happen to match another title in the directory, (unlikely), or will cause a directory entry for a new relative file. The latter is no tragedy.²

Type: GR SAMPLE

The system will locate the file and wait. (Just hit a key and you have the "menu.")

7244 Now you have the MENU and the system is asking you to select a job.



7245 READ EXISTING RECORD

First we'll read a single record. Job # 6 is the one we want.

Type: 6 ← that for "Job # "

Now the system wants a record number.

Type: 2 ← will read record # 2

Follow directions when you get sick of looking at those very creative contents of record # 2.

7246 Back to the menu. Notice that the system tells you what you did last.

Type: 6 ← that's for "Job # "

Type: 10 ← for record #

Whoops! back at the menu because record # 10 does not exist.

Fool around with JOB # 6. By trial and error find out:

- 1) Which record in GR SAMPLE exists but is "not written."³
- 2) What is the last record in GR SAMPLE



² - You can SCRATCH it whenever you want (4500).

³ - (Unless GR SAMPLE has been altered)

7247 This time we'll READ IN SEQUENCE

Type: 5 ← Job #

The system wants a record number to start.

Type: 5 ← to start reading with record # 5

And, of course an ending #

Type: 7 ← to end reading with record # 7

As each record is presented you'll have the option to return to the menu (←) or continue. Keep typing C's.

7248 Experiments with JOB # 5

Rem: One way or another you should be back at the menu. If all else fails rerun the program.

Try these sequences. In each case you'll see an automatic return to the menu. Notice the different messages. Read the foot notes for an explanation.

1) Job # 5, read record # 1 to record # 5 (returns after # 3)²

2) Job # 5, read record # 5 to record # 10 (returns after # 8)³

7249 SECONDARY PROGRAM

Get to the menu and select job # 9 to end the run.

Type: 9 ← this will close the files.

Type: RUN20000

The program will want a file name,

Type: GR SAMPLE

A starting number,

Type: 1

(Continued)

² - Record # 4 has been marked by JOB # 1 - CREATE but it contains no user data.

³ - Record # 8 is the last record in the file. We'd have to use JOB # 2 before we'd be allowed to write record # 9. # 9 does not exist.

7249

(Cont'd)

And last, a number of characters to "get"

Type: 1000

Here is what you'll see if GR SAMPLE is intact. An error message will follow. Ignore it for now.

```

GR SAMPLE
SMITH+#####JOHN+#####111 11 111
1+11+M+1.11+11 11 67+111 1111+8+JONES+##
#####MARY+#####222 22 2222+12+F+2
,22+2 FEB 67+222 2222+TRANSFER+IMAGINATI
ON+#####NONE+#####333 33 3333+3+3+3.
33+33 33 33+333 3333+DULL+#+NUMBER 5+###
#####FIVE+#####555 55 5555+5+5+5.00+
MAY 5+#####555 5555+NONE+NUMBER 6+#####
SIX+#####666 66 6666+12+N+6.66+6 JN
69+666 6666+NONE+SAMPLE+#####RECOR
D+#####123 12 1234+XX+X+X.XX+XX XX+###X
XX XXXX+XXXXXXXXXX+ANOTHER+#####SAMPL
E+#####NA+#####9+N+4.00+31FEB65+1
23 4567+123456789+

```

7250

USER INSTRUCTIONS - General

Rem: If you have not completed the previous section, do so now.

Rem: In this section we'll explore the rest of the jobs in "GEN REL" by creating and writing another sample file.

Rem: If this is your first time through these you'll be creating a file called "PROJECT." Special notes are in clouds.

Type: RUN

Examine the file format to be sure it meets your needs. If so, continue. If not adapt (7270)

The system now wants the file name. Think before typing.

Type: (Name of file)

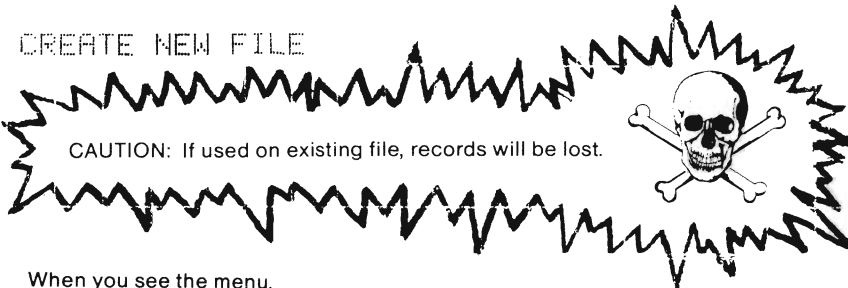
PROJECT

I assume your system is up, Wedge active, and GEN REL loaded. You may want to change to your test disk.

It does if you're doing the "PROJECT"

If you type the name of an existing file it will be opened here.
→
new file - a directory entry will be created for it.

7251 CREATE NEW FILE



When you see the menu,

Type: 1

You will see the job title and a warning. In general if you are not sure about the contents of the file it would be best to use the escape option and check by reading.

If you chose to continue you now are told that the directory entry was already created.

GEN REL is asking for the "anticipated number of records."

Allow a few more than you think you'll need.

Type: () whatever

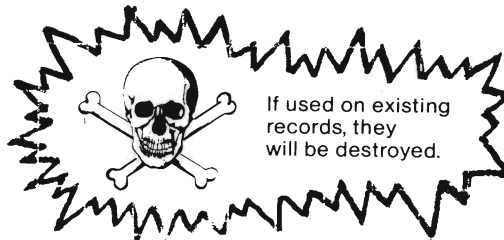
6 for "PROJECT"

You have one more chance to escape. NOTE Chosen number for future reference. If you chose to continue, you're back at the menu.

Continue if you are doing the "project"

Project - use 9 to end run as in (7249) run 20000. Notice 6 stars followed by "50 Record Not Present." Those markers are there to protect the user as described earlier. (7220)

7252 EXPAND FILE ²



Type: 2 Job number

You will see the job Title and a warning, in general. (7251)

If you choose to continue, GEN REL asks for the current last record numbers.

Type: () whatever

6 for "PROJECT"

Now GEN REL wants to know how many records to add.

Type: () whatever

4 for "PROJECT"

You must type something here. There is an escape option coming so if you don't know, type anything for now.

(Continued)

² - If you did the last "Project," just reRUN, "PROJECT" for title.

7252 (Cont'd)

When the system is finished marking those records, GEN REL returns you to the menu.

Project: Use 9 for job number to end run.

Type: RUN20000 to run secondary program.



Notice 10 stars and possibly some " "s followed by 50-record not present.

PROJECT
RN= 1 L= 1000
* * * * *

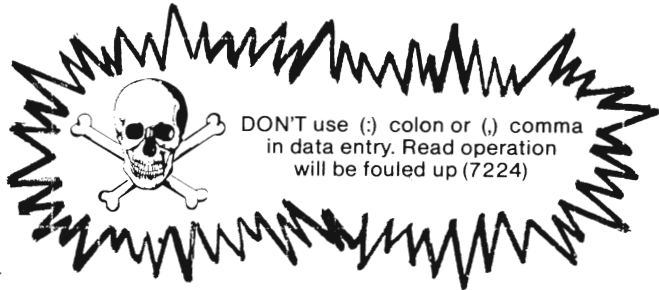
← after JOB # 2

PROJECT
RN= 1 L= 1000
* * * * *

← before JOB # 2

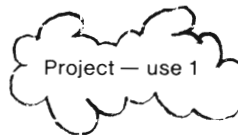
7253 WRITE IN SEQUENCE

Type: 3 ← for job number.



GEN REL asks for a starting record number.

Type: () ← whatever your starting number



GEN REL asks for an ending record number.

Type: () ← whatever your ending number



Your first number appears on the screen. If it has already been written you will see a message. In any case, you can escape if you choose.

If you choose to continue, you'll see a prompt followed by the number of spaces allowed for user input (field length). GEN REL will truncate if you type more than the allowed number. There is no problem if you type fewer. If you simply hit return you'll get either a dash or left overs from preceding records. If you don't want to enter contents, it's best to use some symbol other than (:), or (,).

Type: (Your data)



(Continued)

7253 (Cont'd)

As you continue to enter data (no escape option until the record is completely written) some of the prompts will give a format. You are not restricted to that format.

When the record is completely written, you will be given the escape option.

When the sequence is complete or GEN REL finds a non-existent (not marked by jobs 1 or 2) you'll be returned to the menu.

Project: Play with READ options jobs 5 and 6.
Run 20000 for secondary program as in earlier sections.
Notice the *'s for the unwritten records.
Examine the directory if your wedge is active or you don't mind reloading.

7254 WRITE INDIVIDUAL RECORD

Type: 4 ← Job #

GEN REL wants number of the record you wish to write.

Type: () ← whatever.

Project: see below

Your record is presented for writing is in the previous job # 3 (7253)

GEN REL will wait for you to check your data and return you to the menu. If you notice an error, especially a small one the next job may be EDIT. With that job you'll be able to fix it easily.

Project:

Type: 6 ← to write record # 6

Type: C ← to continue

Type: (data)

when you land back at the menu select this job again.

Type: 4 ← Job #

Type: 3 ← Record #

Notice the message that the record is written. You can either rewrite it or go back to the menu.

When you land back at the menu, select this job again.

Type: 4 ← Job #

Type: 13 ← Record #

Notice you got back to the menu. #13 was not marked by jobs 1 or 2. If you have an overpowering desire to write #13 you must do Job #2.

7255 READ IN SEQUENCE (see first run 7247)

7256 READ EXISTING RECORD (see first run 7245)

7257 EDIT EXISTING RECORD

Type: 7 ← Job #

Type: () ← Record #

Project: Use a number you know you've written

GEN REL will show you the contents of your record and present you with the edit menu.

0 will return you to the menu. This is the way out.

10 will present all fields for rewriting, show you the new contents.

1 to 9 allow you to change individual fields. After each field change you may select another field, (1 - 9), or 0 to return to "the" menu.

7258 CHANGE FILE

Type: 8

GEN REL is back at File Name

At this point you can change disks if you like. Remember, when you enter a file name that is not on the disk in the drive, a directory entry for that file name will be created. You can always scratch it later (4500).

Project: Enter new file name, end run and examine disk directory. Notice 0 blocks used. Only the directory entry has been made.

7259 END RUN

Type: 9

Projects: Run secondary program and examine your file.

- Run main program again and test expand or create file. You'll destroy data but so what!
- Expand past the end of your file. i.e. If you've created records 1 through 8, expand using 12 as your last record and add a couple more. Run secondary program to see what your file looks like. **tf** 's are explained in (7321).

Rem: The file in GEN REL as structured is probably of little use to you personally. The next few sections will explain how to adapt the program to suit yourself.

7260 RECORD STRUCTURE

Rem: Much of the information in this section is of little or no concern to the casual user. If your goal is simply to adapt "GEN REL" to your own needs, you do not need total mastery here.

Rem: If your goal is to write your own relative file programs, it would be wise to put some time into this section.

7261

FIELD #	DATA CATEGORY	FIELD LENGTH		RECORD LENGTH	STARTING POSITION
		INPUT	ACTUAL		
1	LAST NAME	16	17	17	1
2	FIRST NAME	12	13	30	18
3	SOCIAL SEC #	11	12	42	31
4	CLASS (9 - 12)	2	3	45	43
5	SEX (M/F)	1	2	47	46
6	AVE (X, XX)	4	5	52	48
7	D.O.B. (XX-XX-XX)	8	9	61	53
8	PHONE#(XXX-XXXX)	8	9	70	62
9	COMMENT	9	10	80	71

(a) (b) (c) (d) (e)

(a) and (b) are determined by the programmer or user adapter based on the purpose of the file. Details and record planning tables are in the next section (7270)

(c) - (e) are computed by "GEN REL"

(c) One byte must be allowed for "separations" (5400) for each data entry.

$$ACTUAL = INPUT + 1$$

(d) The last entry is the record length for this file.
 $RECORD\ LENGTH = PREVIOUS\ RECORD\ LENGTH + ACTUAL\ FIELD\ LENGTH\ (this\ entry)$

(e) These numbers will be used to "position" pointer
 $STARTING\ POSITION = PREVIOUS\ RECORD\ LENGTH + 1$

7262

General Structure²

POSITION *	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
CONTENTS	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	←	--	--	--

POSITION	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
CONTENTS	--	--	--	--	--	--	--	--	--	←	--	--	--	--	--	--	--	--	--	--

POSITION	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
CONTENTS	--	←	--	--	←	--	←	--	--	--	←	--	--	--	--	--	--	--	--	--

POSITION	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
CONTENTS	←	--	--	--	--	--	--	--	--	←	--	--	--	--	--	--	--	--	--	←

“ ← ” denotes end of field in this figure only. (not on disk).

² - These Tables give a computer's eye view similar to that given by the secondary program (7249).

7263 Sample containing data.²

POSITION	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
CONTENTS	S	M	I	T	H	←	-	-	-	-	-	-	-	-	-	-	-	-	J	O	H

POSITION	1	2	3	4	5	6	7	8	9	30	1	2	3	4	5	6	7	8	9	40
CONTENTS	N	←	-	-	-	-	-	-	-	-	1	2	3	-	4	5	-	6	7	8

POSITION	1	2	3	4	5	6	7	8	9	50	1	2	3	4	5	6	7	8	9	60
CONTENTS	9	←	1	0	←	M	←	3	.	0	0	←	1	-	1	1	-	6	8	←

POSITION	1	2	3	4	5	6	7	8	9	70	1	2	3	4	5	6	7	8	9	80
CONTENTS	-	5	5	5	-	5	5	5	5	←	N	O	N	E	←	-	-	-	-	-

DATA ENTERED

1. SMITH
2. JOHN
3. 123-45-6789
4. 10
5. M
6. 3.00
7. 1-11-68
8. 555-5555
9. NONE

" ← " shows location of separator. Unused bytes contain nulls.

7264 SPANNING.³ (record length = 80 bytes)

RECORD #	1	2	3	4	4	5	6	7	7	8	9	10	10	11	...	
bytes	80	80	80	14	66	80	80	28	52	80	80	42	38	80	...	
data bytes	254				254				254				254			
DATA SECTOR	1				2				3				4			

² - These tables give a computer's eye view similar to that given by the Secondary Program (7249).

³ - Spanning is not a programmers problem. DOS takes care of the details.

7270 **USER ADAPPTIONS**

7271 General Flow

- 1st: Decide on the purpose of your file.
- 2nd: Determine Records Contents - Data Categories (7274)
- 3rd: Set Field Lengths (7274)
- 4th: LOAD "GEN REL"
- 5th: List 900 - 999.
- 6th: Alter data lines
- 7th: Remove the REM from program line 115 if you need more than 10 fields.
- 8th: Run "GEN REL"
- 9th: SAVE if FILE FORMAT acceptable, *new program* under a *new name*.
- 10th: Create your file.
- 11th: Use your new file with your new program



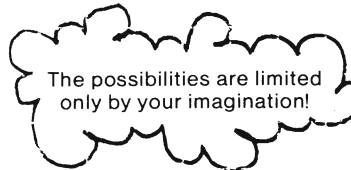
7272 Your file can contain anything you want it to (within limits). Here are but a few samples:

MAILING LIST
 1. Last Name
 2. First Name
 3. Street Address
 4. City
 5. State
 6. Zip Code
 7. Comment

CHECK REGISTER
 1. Date
 2. To:
 3. For:
 4. Amount:
 5. Deductable (Y/N)

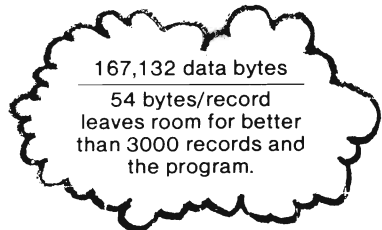
ARTICLES
 1. Title
 2. Magazine
 3. Issue
 4. Category
 5. Level
 6. Author

GRADE BOOK
 1. Last Name
 2. First Name
 3. Test 1
 4. Test 2
 etc., etc., ...
 52. Test 50



7273 Setting Field lengths will require a little thought. Keep in mind: Total Record length is limited to 254 bytes and that you must leave room for separations.

ex. CHECK REGISTER	user input	
1. Date:	8 + 1 = 9	actual field length
2. To: (a name of sorts)	16 + 1 = 17	
3. For: (memo)	16 + 1 = 17	
4. Amount:	8 + 1 = 9	
5. Deductable: (Y/N)	1 + 1 = 2	
	49 + 5 =	54 record length



If you need a lot of records in the file, record length should be kept to a minimum. The date entry could be cut by deleting the year and including that in the file name.

(Continued)

7273 (Cont'd)

Record Length	Max record # ³
50	3342
100	1671
200	835
254	658

167,132 data bytes

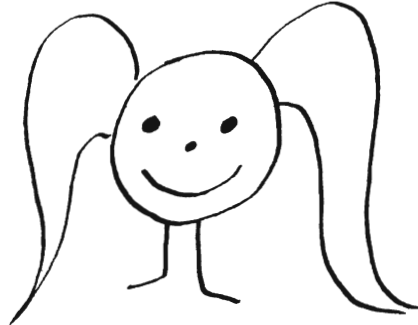
ex. GRADE BOOK²

	16 + 1	=	17		
1	LAST NAME		12 + 1	=	13
2	FIRST NAME		3 + 1	=	4
3	GRADE 1				
4	GRADE 2				
	"				
	"				
	"				
	"				
	"				
51	GRADE 49				
52	GRADE 50				
			3 + 1	=	4
			<u>178 + 52</u>	=	<u>150</u> record length

150 { 50 } 200

As your skill increases work on efficiency. Long files can take a long time to read and to write. Saving 5 bytes per record saves 500 bytes, (2+ blocks), in a file of 100 records!

You may want to make some copies of the following table...



² - If you want to do something like this you will want to alter format in READ sequences(7333) and file format presentation (7312).
³ - relative file data bytes available divided by record length.

7274 RECORD PLANNING TABLE

FIELD NUMBER	DATA CATEGORY	FIELD LENGTH		RECORD LENGTH*	STARTING POSITION *
		INPUT	ACTUAL*		
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					

*The casual user need not worry about these columns. "GEN REL" will do the computations and abort the run if the record gets too long.

7275 If your record is planned (7420), your wedge active (1230) and "GEN REL" in computer memory (7241),

Type: LIST900-999



7276 Alter DATA statements

NOTE: The first data entry must be the total number of fields in your record. i.e. Use the field number of your last data category.

910 DATA9 ← Number of Fields

All the other data statements contain a user prompt (string) followed by the length of that field (number).

```

911 DATA LAST NAME, 16
912 DATA FIRST NAME, 12
913 DATA SOC SEC #, 11
914 DATA CLASS(9-12), 2
915 DATA SEX(M/F), 1
916 DATA AVE(X, XX), 4
917 DATA D.O.B. (XX-XX-XX), 8
918 DATA PHONE#(XXX-XXXX), 8
919 DATA COMMENT, 9
    
```

PROMPT

User prompts will appear on the screen. You can do just about anything you want with these. Keep them under 20 characters or you'll want to change format in READ RECORD sequences in the program (see program line 4060) and possibly the FILE FORMAT presentation (see program line 230).

Field lengths must be positive integers and must follow their prompts.

Note: If you do not write over all of these data lines, it would be good practice to delete the ones that do not apply to your program.

Note: You may, of course, "crunch" your data statements. I simply found it easier to deal with them line-by-line. If you choose to "crunch" be sure to include the number of fields FIRST.

Format for "crunched" data lines:

```

...DATA number of fields, prompt, length, prompt, length, ...
...DATA prompt, length, ..., prompt, length
    
```

7277 To Check Format.

Type: RUN

Your format should appear on the screen. If your record length exceeds 254 the run is aborted automatically. If your new format pleases you, you're ready to go to work. If not, chose the "escape" option and fiddle.

NOTE: DO NOT attempt to use files created with "GEN REL" with this new program.

7278 Choose a new name for your new programs and SAVE it. You can put it on "Friendly Floppy" if there's room.

SAVE "NEW NAME", B normal save - no wedge

+NEW NAME with wedge

7279 You should now be ready to create your own files with your own record structure. Good Luck.

7280 **UNWANTED RELATIVE FILES**

The only way to dispose of unwanted relative files is to use the SCRATCH COMMAND. (4500)



7300 "GEN REL" - Detailed Explanation

Rem: Although not necessary, it would be helpful to have your system up and GEN REL loaded as you can list portions and work with them as you go. If you have not completed the previous section, First Run (7240) at the very least, it would be wise to do so.

References to text lines are enclosed in parentheses, (). References to program line numbers are not.

Before getting into the program itself a few, hopefully handy references are included.



(1000) Text line
1000 Program line

7301 Variable List.

MAIN PROGRAM

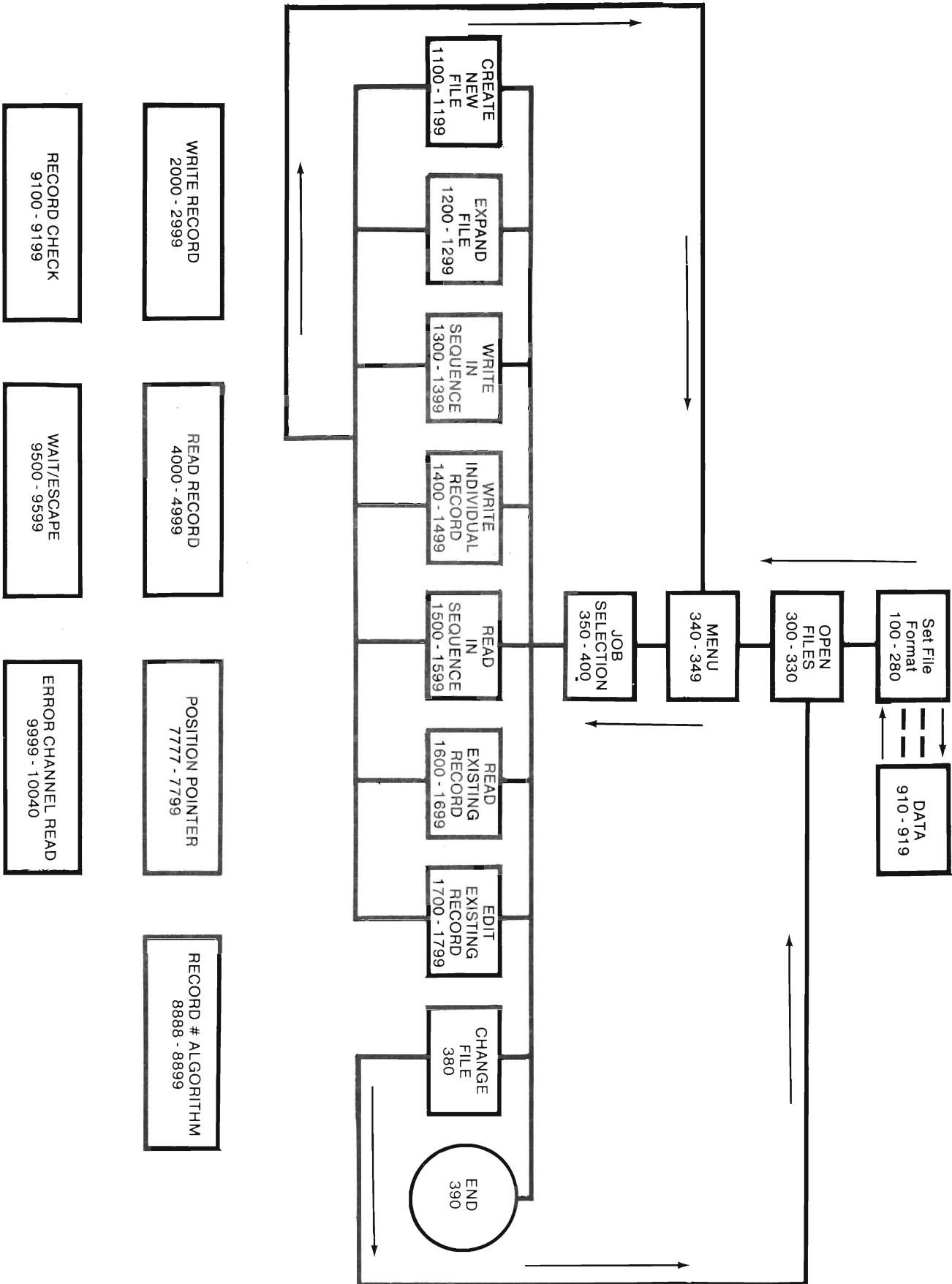
D = number of fields in each record (data statement)
 DC#() = data categories for prompts (data statement)
 L() = field lengths (data statements)
 P() = starting positions of fields (calculated)
 L = record length (calculated)
 I = counter for FOR-NEXT loops
 FF\$ = file name (user input)
 K\$ = string for "wait" sequences (user input)
 M = job number from menu (user input)
 RN = record number (user input or counters)
 N = number of records protected when file created (user input)
 N = field number to edit (user input)
 P = position in record (calculated)
 D#() = user data (user input or input from file)
 S = last record for expansion sequence (user input)
 A = number of records to add in expansion sequence (user input)
 C\$ = for record check sequences (input from file)
 HI = high byte of record number (calculated)
 LO = lo byte of record number (calculated)
 EN = error number (input from command channel)
 EM\$ = error message (input from command channel)
 ET = error track (input from command channel)
 ES = error sector (input from command channel)

SECONDARY PROGRAM

L = number of characters to GET# (user input)
 G\$ = character (input from file)

MAIN PROGRAM - GENERAL FLOW AND SUBROUTINES

Rem: Numbers are program lines to LIST.



7310 INITIAL SET UP

7311 SET FILE FORMAT

```

100 REM *** GEN REL ***
110 READD
115 REM DIM DC$(D),L(D),P(D),D$(D)
120 FORI=1TOD
125 READDC$(I),L(I)
130 P(I)=L(I-1)+P(I-1)+1
135 L=P(I)+L(I)
140 D$(I)="-"
145 NEXT
    
```

Presented "out of sequence" for your convenience. These follow 400.

```

910 DATA9
911 DATA LAST NAME,16
912 DATA FIRST NAME,12
913 DATA SOC SEC #,11
914 DATA CLASS(9-12),2
915 DATA SEX(M/F),1
916 DATA AVE(X,XX),4
917 DATA D.O.B.(XX-XX-XX),8
918 DATA PHONE#(XXX-XXXX),8
919 DATA COMMENT,9
    
```

110 Reads 910
 D = 9 since there are to be 9 data fields.
 If the user chooses to alter the number of fields (7270), 910 must be changed accordingly.

115 The **REM** must be removed if there are more than 10 fields.

120 - 145 125 reads 911 to 919, (Data Category and Field Length) 130 and 135 perform the calculations shown below. See "record structure" (7260) for other details.
 140 fills the user data list with a character to prevent the system from tying up if the user fails to enter data on the first write sequence.

VALUES during RUN				
I	DC\$(I)	L(I)	P(I)=L(I-1)+F(I-1)+1	L
1	LAST NAME	16	1 = 0 + 0 + 1	17
2	FIRST NAME	12	18 = 16 + 1 + 1	30
3	SOC. SEC. #	11	31 = 12 + 18 + 1	42
4	CLASS(9-12)	2	43 = 11 + 31 + 1	45
5	SEX (M/F)	1	46 = 2 + 43 + 1	47
6	AVE (X,XX)	4	48 = 1 + 46 + 1	52
7	D.O.B. (XX-XX-XX)	8	53 = 4 + 48 + 1	61
8	PHONE # (XXX-XXXX)	8	62 = 8 + 53 + 1	70
9	COMMENT	9	71 = 8 + 62 +	(80)

7312 FILE FORMAT PRESENTATION

```

200 PRINT"FILE FORMAT"
210 PRINT"CODE"TAB(20)"LENGTH  POSITION"
220 FORI=1TOD
230 PRINTDC$(I)TAB(20)L(I)TAB(30)P(I)
240 NEXT
250 PRINT"RECORD LENGTH",L"
255 IFL>254THENPRINT"TOO LONG":END
260 PRINT"TYPE - C TO CONTINUE / ← ESCAPE PROGRAM"
265 GETK$:IFK$=""THEN265
270 IFK$="+"THENEND
280 IFK$<"C"THEN265

```

200 - 250 Simply print results of the preceeding lines to the screen.

255 Aborts the program if user adaptations exceed the maximum allowed record length for relative files. This line is not active with the data lines used in GEN REL and is intended only to protect the user adaptor(7270)

260 - 280 Escape option for user

NOTE: All of these lines, 200 - 280, are strictly basic BASIC. Only the "254" in line 255 requires knowledge of relative files. If you are having trouble, you need to consult a reference on BASIC.

7313 OPEN FILES

```
300 OPEN15,8,15  opens the command channel.
```

```
310 INPUT"FILE NAME";FF$  simply asks for user to give name of file to work with.
```

This file name remains constant unless the user elects to CHANGE FILE (Job # 8). When # 8 is selected the system is sent to this line of the program.

```
320 OPEN3,8,3,FF$+",L,"+CHR$(L)
```

7313 OPEN FILES (Cont'd)

Note: Please accept my apologies for the two L's in this statement. The following sample statement is presented for contrast.

```
OPEN 3,8,3,"FILE X,L," + CHR$(58)
```

Record Length = 58

file TYPE. Will show as REL in directory.

file name of course

Refer to (7141) for function.

```
321 GOSUB9999 "Error Channel Read"
```

Note: Subroutines are described in detail in (7338). At this point in the program nothing too awful can happen. If active at all, it's usually "No disk in drive." "File not found" will not occur since if the file does not exist, it will have its directory entry created in 320. "File type mismatch" will occur if the user inputs the name of existing sequential file or the name of a program.

```
325 PRINT"ANY KEY TO CONTINUE"
327 GETK$:IFK#=""THEN327
330 PRINT"J"
```

Simply waits for the user and clears screen for menu.

7314 The Menu

```
340 PRINTTAB(18)"MENU":PRINT"FILE:  "FF$"
341 PRINT" 1 - CREATE NEW FILE
342 PRINT" 2 - EXPAND FILE
343 PRINT" 3 - WRITE IN SEQUENCE
344 PRINT" 4 - WRITE INDIVIDUAL RECORD
345 PRINT" 5 - READ IN SEQUENCE
346 PRINT" 6 - READ EXISTING RECORD
347 PRINT" 7 - EDIT EXISTING RECORD
348 PRINT" 8 - CHANGE FILE
349 PRINT" 9 - END RUN
```

Note: Job # 1 is described in (7321) program lines 1100 - 1199
 Job # 2 is described in (7322) program lines 1200 - 1299
 Job # 3 is described in (7323) program lines 1300 - 1399
 Job # 4 is described in (7324) program lines 1400 - 1499
 Job # 5 is described in (7325) program lines 1500 - 1599
 Job # 6 is described in (7326) program lines 1600 - 1699
 Job # 7 is described in (7327) program lines 1700 - 1799

All phases of the program end up here, line 340, except unscheduled errors.

7315 **Job Selection**

```

350 IFM>0THENPRINT"LAST JOB: "M
355 PRINT"FILE: "FF$, REC # "RN"
360 INPUT"JOB #";M
370 IFM<1ORM>9THEN360

```

M = Job number

350 reminds user of last job
 355 reminds user of file name and last record accessed
 360 simply asks for the next task
 370 is for user protection

7316 **Funnels**

```

380 IFM=8THENCLOSE3:GOTO310

```

will ask for file name, open it and land at the menu.

↑
CHANGE FILE

↑
necessary since another file will be opened under another name using
the same file number and data channel.

```

390 IFM=9THENCLOSE3:CLOSE15:END

```

← except for unscheduled errors, the program ends here.

↑
END RUN

↑
data channel
file number

↑
command channel file number

(5320) for details on CLOSE

```

400 ONMGO TO 1100,1200,1300,1400,1500,1600,1700

```

↑
Job Number

see note (7314)

7320 **JOBS**

Rem: You may wish to review job descriptions in (7220). (7221) corresponds to (7321) etc.

```

7321 1100 REM ** CREATE NEW FILE **

```

```

1110 PRINT"CREATE: "FF$
1120 PRINT"WILL DESTROY DATA IF USED ON OLD FILE"
1121 GOSUB9500

```

↑
"wait" allow user to escape. (7337)

7321

(Cont'd)

```
1130 PRINT"␣DIRECTORY ENTRY CREATED"
```

The lines above are for user information and protection.

Notice: The directory entry of a new file was created in 320. If the file already existed, the communication link was opened and no harm to file contents occurred.

```
1135 PRINT"␣ANTICIPATED NUMBER OF RECORDS":INPUTN
```

N records will be marked

```
1140 GOSUB9500    "Wait" (7337) ← user's last chance to escape.
```

```
1160 P=1    will be used in "position pointer" routine. Pointer will be set to 1st position of each record. 1185 actually places the mark.
```

1165 FORI=1TON	from 1135
1170 RN=I	RN = record number
1175 GOSUB8888	"Record Algorithm"(7335) List 8888 - 8899
1180 GOSUB7777	"Position Pointer" (7334) List 7777 - 7799
1185 PRINT#3,"*"	places marks in data file
1186 GOSUB9999	"error channel read" (7338)
1190 NEXT	List 9999 - 10040



The real work is done in these subroutines

```
1199 GOTO340    The Menu
```

Note: This procedure is not an absolute requirement for relative files. The marks, *, will be used for user protection in subroutine 9800. (7336)

In general, it would suffice to position the pointer to record # N. Intermediate records would be marked by the system with binary 1's in the first position. $11111112 = 255_{10}$ and corresponds to $\text{CHR}\$(255) = "\pi"$. That's why you sometimes see π 's when you run the secondary program. DOS seems to mark an entire sector once the relative file enters that sector.

7322 1200 REM ** EXPAND FILE **

Rem: This "job" does essentially the same thing as the preceding. The program structure of 1200 - 1299 parallels that of 1100 - 1199 with a few exceptions. Some general techniques and principles are described so don't overlook this section.

```
1210 PRINT"***EXPAND***: "FF$
1220 PRINT"***WILL DESTROY DATA IF USED ON OLD FILE***"
```

A line 1221 GOSUB 9500 should probably be added here to save the user from anxiety attacks.²

```
1230 PRINT"CURRENT LAST RECORD #":INPUTS
1235 PRINT"NUMBER OF RECORDS TO ADD":INPUTA
```

S unfortunately depends strictly on user input. This is one of the few dangerous spots in the program since it can cause destruction of data. Once the "*" is placed in the 1st position of the record, previous contents from position 2 to the end of the record are gone. Individual field writing can be dangerous.

e.g. Say we have 9 fields. Writing field # 8 destroys contents of # 9, but leaves 1 through 7 intact. Writing field # 4 destroys # 5 through # 9 but leaves # 1 through 3 intact.

Notice: These are field numbers not record numbers. It is perfectly safe to write record # 13. All other records are left untouched if they already exist.

1240 GOSUB9500	←	"WAIT" LIST 9500 - 9599
1260 P=1	←	for "position pointer" like 1160
1265 FORI=S+1TOS+A	←	e.g. if S = 15 and A = 7, I = 16 to 22 ³
1270 RN=I	←	RN = record number
1275 GOSUB8888	←	"POSITION POINTER" LIST 8888 - 8899
1280 GOSUB7777	←	"RECORD # ALGORITHM" LIST 7777 - 7799
1285 PRINT#3,"*"	←	places mark in data file
1286 GOSUB9999	←	"ERROR CHANNEL READ" LIST 9999 - 10040
1290 NEXT		
1299 GOTO340	←	MENU

7323 - 7326 Rem: Notice Parallel Structure

² - Frankly I'd stop and put that in but if I keep fussing with this program, this book will never get finished.

³ - If you find this as awkward as I do, you may wish to change in a manner similar to that in 1300 and 1500.

```

7323 1300 REM ** WRITE IN SEQUENCE **
      1305 PRINT"WRITE IN SEQUENCE STARTING WITH"
      1310 INPUT"RECORD #";S
      1315 INPUT"WRITE TO #";A
      1320 FORJ=STOR
      1330 RN=J
      1340 GOSUB2000 ← "WRITE RECORD"
      1350 GOSUB9500 ← "WAIT" 9500 - 9599
      1360 NEXTJ
      1399 GOTO340 ← MENU
  
```

"WRITE RECORD" LIST 2000 - 2999
accomplishing major writing tasks here
and in 1420. (7332)

```

7324 1400 REM ** WRITE INDIVIDUAL RECORD **
      1410 INPUT"RECORD #";RN
      1420 GOSUB2000 ← "WRITE RECORD"
      1430 PRINT"ANY KEY FOR MENU"
      1435 GETK$:IFK$=""THEN1435
      1499 GOTO340 ← MENU
  
```

```

7325 1500 REM ** READ IN SEQUENCE **
      1505 PRINT"READ IN SEQUENCE STARTING WITH"
      1510 INPUT"RECORD #";S
      1515 INPUT"READ TO #";A
      1520 FORJ=STOR
      1530 RN=J
      1540 GOSUB4000 ← "READ RECORD"
      1550 GOSUB9500 ← "WAIT" 9500 - 9599
      1560 NEXTJ
      1599 PRINT"J":GOTO340 ← MENU
  
```

"READ RECORD" LIST 4000 - 4999
accomplishes read and display tasks
here and in 1620. (7333)

```

7326 1600 REM ** READ INDIVIDUAL RECORD **
      1610 INPUT"RECORD #";RN
      1620 GOSUB4000 ← "READ RECORD"
      1630 PRINT"ANY KEY FOR MENU"
      1635 GETK$:IFK$=""THEN1635
      1699 GOTO340 ← MENU
  
```

```

7327 1700 REM ** EDIT RECORD **

1710 INPUT "RECORD #";RN
1715 GOSUB4000 ← To show current record contents. if none, user returned to menu.

```

↑
"READ RECORD" LIST 4000 - 4999 (7333)

```

1720 PRINT "MTYPE: 0 - NO CHANGE
1721 PRINT "      10 - CHANGE ALL
1722 PRINT "      1 TO 9 - FIELD CHANGE
1725 INPUTN: IFN<00RN>10THEN1725

```

} "Edit" menu and job selection.
Choices 1 - 9 send us to 1740

```

1730 IFN=0THEN340      MENU

```



```

1735 IFN=10THENGOSUB2030:GOTO1715

```

↑
Jumps to WRITE RECORD
LIST 2000 - 2999
Simply writes a new record. (7332)

← Allows user to check newly entered data.

```

1740 PRINTD$(N)      Data category as read in 125
1745 INPUTD$(N)      User data. Note this affects only data in computer memory list not
                    data on disk. D$(N) was read in 4000 - 4999

```

```

1750 D$(N)=LEFT$(D$(N),L(N))  protects from overwriting field like 2055 (7332)

```

↑
field length as read in 125

```

1755 PRINT "ANOTHER FIELD?(0-9)":INPUTN
1760 IFN=0THEN1775
1765 IFN<10RN>9THEN1755
1770 GOTO1740

```

allows user to change another field

The following, 1775 - 1795, re-write the entire record, reflecting the changes in the list, D\$(I). (See note under 1235 in (7322).)

```

1775 FORI=1TOD      ← as read in 110 from 910
1780 P=P(I)        ← as computed in 130 for "POSITION POINTER"
1785 GOSUB7777     ← "POSITION POINTER" LIST 7777 - 77992
1790 PRINT#3,D$(I) ← places data in file. 3 is the fn of the data file
1791 GOSUB9999     ← "ERROR CHANNEL READ" LIST 9999 - 10040.
1795 NEXT          Should find an "OK" unless something weird has
                    happened.

```

```

1799 GOTO1715      allows user to check newly entered data

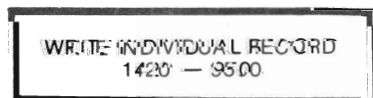
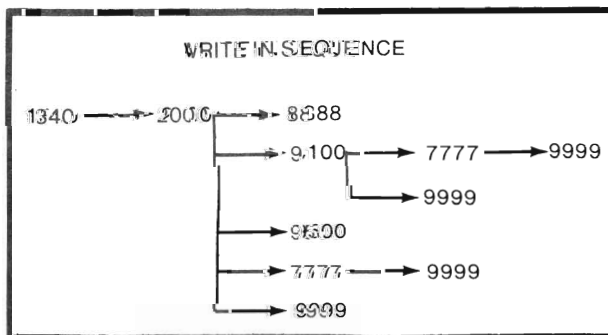
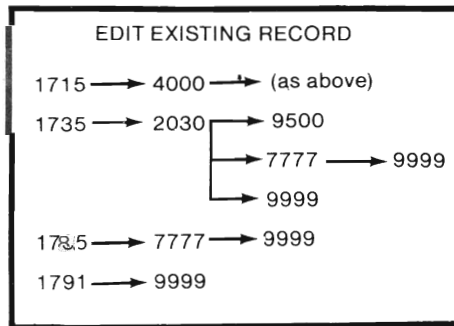
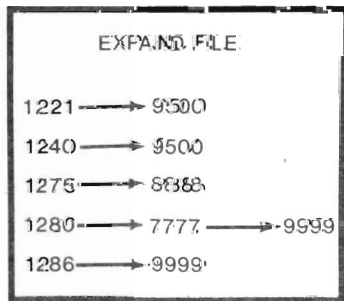
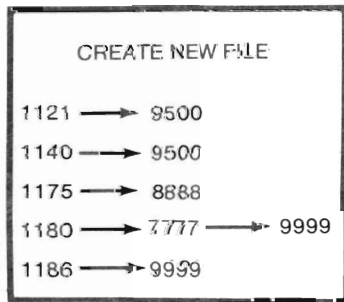
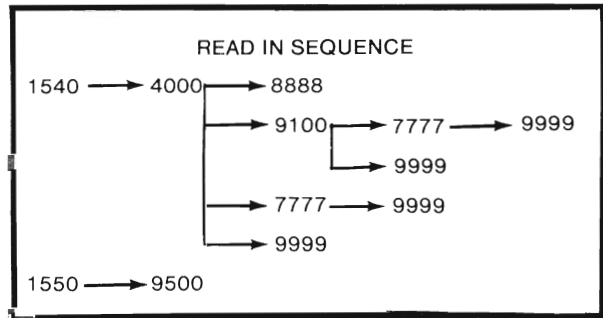
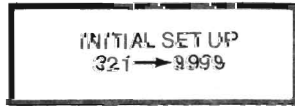
```

² - "RECORD # ALGORITHM" LIST 888 - 8899 has already been done in 4010 from 1715. (7335)

7330 1999 REM *** SUBROUTINES ***

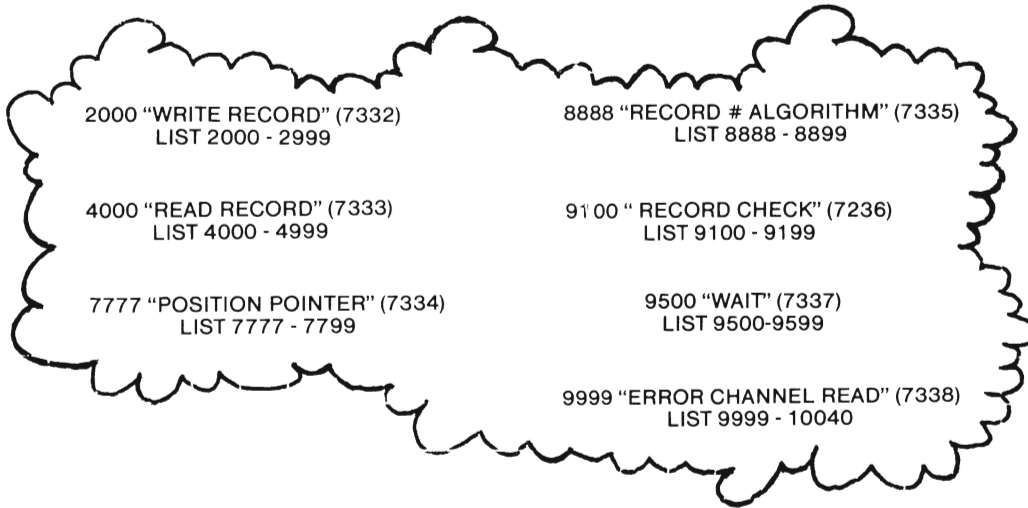
Rem: This Tracer is presented to help you figure out where you've been and/or are going. If you feel like you are going around in circles, relax. As you can see, the seven little subroutines are nested, as often as four deep. Be patient.

7331 SUB TRACER



(Continued)

7331 (Cont'd)



7332 2000 REM ** WRITE RECORD **

This sequence performs actual record writing with the aid of other subroutines. It is encountered from Job #'s 3, 4 and 7.

2005 PRINT "RECORD #"
Record # from user input Jobs 4 & 7 or computed in Job # 3.

2010 GOSUB 8888 "RECORD # ALGORITHM" LIST 8888 - 8899 (7335)

2011 GOSUB 9100 ← "RECORD CHECK" LIST 9100 - 9199 (7336)
 2020 IFC#="*" THEN 2029 ← as marked by job # 1 or job # 2
 2025 PRINT " HAS BEEN WRITTEN" ← Warning if record contains user data
 2029 GOSUB 9500 ← "WAIT" LIST 9500 - 9599. Allows user to over-write if desired

2030 FOR I=1 TO D as read in 110 from 910

2035 P=P(I) used in "POSITION POINTER" as computed in 130 at each pass 2060 will position the pointer to the beginning of the field to be written.

2040 PRINT D\$(I) ["L(I) SPACES"]
 ↑ data category heading ↑ field length
 as read in 125 from 911 - 919

2050 INPUT D\$(I) user data list. If user hits return and thus fails to enter data, D\$(I) keeps its current value. 140 puts -'s into these positions. The return key functions as usual and is not recognized as input. Colons (:) and commas (,) foul up read operations (7224) & (5400)

7332 (Cont'd)

2055 D\$(I)=LEFT\$(D\$(I),L(I)) Tuncates if string input was longer than the allowed length. A real mess results when fields are overwritten. i.e., If data separators are destroyed, read operations will not work properly.

2060 GOSUB7777 "POSITION POINTER" LIST 7777 - 7799 (7334) uses values from 2005, 8888, 2035 to position to the beginning of the field I in the record # specified, RN.

2070 PRINT#3,D\$(I) Puts the data into the buffer for transfer to disk when buffer is filled or file closed (5320). A data separator is placed automatically. (5400).

2071 GOSUB9999 "ERROR CHANNEL READ" LIST 9999 -10040
2080 NEXT

2099 RETURN to 1340 or 1420 or 1735

7333 4000 REM ** READ RECORD **

This sequence performs the actual record reading with the aid of other subroutines. It is encountered in Jobs #'s 5, 6 and 7

4010 GOSUB8888 "RECORD # ALGORITHM" LIST 8888 - 8899. (7335) Use RN from 1530, 1610 or 1710.

4011 GOSUB9100 "RECORD CHECK" LIST 9100 - 9199 (7336)

4020 IFC\$="*"THENPRINT"#####RN" NOT WRITTEN":GOTO340 ← MENU

When the program encounters an unwritten record, this line prevents a "STRING TOO LONG" BASIC error message and an aborted run. Were that to occur and if the user failed to close files manually, data could be lost.

4025 PRINT"#####FF#TAB(20)#####RN" user information

4030 FORI=1TOD
4035 P=P(I)
4040 GOSUB7777
4050 INPUT#3,D\$(I) ← receives a string from the file
4051 GOSUB9999
4060 PRINTI" "DC\$(I)TAB(22)D\$(I)
4070 NEXT
4099 RETURN ← to 1540 or 1620 or 1715

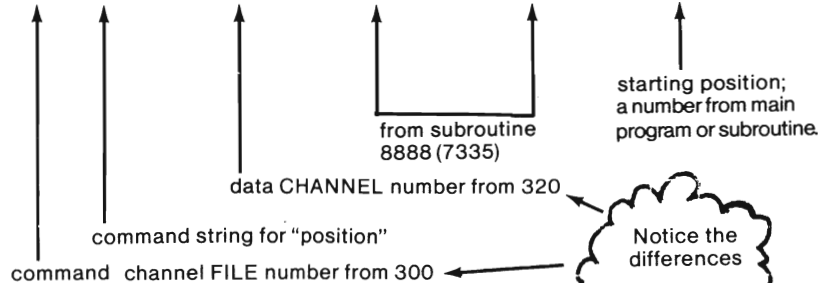
these lines are similar to 2030 -2099 If you are considering changing to more than 20 fields, (7270) you'll probably want to alter record presentation format. This is the best place for that.

7334 7777 REM *** POSITION POINTER ***

data file numbers line 320

This routine is encountered prior to all PRINT # 3, INPUT # 3, and GET # 3 commands. (7143) contains general format.

7780 PRINT#15, "P"CHR\$(3)CHR\$(LO)CHR\$(HI)CHR\$(P)



Note: Again I apologize for the two P's. Be sure to note the differences.

7781 GOSUB9999 "ERROR CHANNEL READ" LIST 9999 - 10040 (7338)

7799 RETURN to 1180 or 1280 or 1785 or 2060 or 4040 or 9120

7335 8888 REM *** RECORD # ALGORITHM ***

8890 HI=INT(RN/256) HIGH & LOW BYTES of RN (record number)

8891 LO=RN-HI*256 for use in SUBROUTINE 7777

8899 RETURN to 1175 or 1275 or 2010 or 4010

A single byte can handle at most 256 (0 to 255) numbers. Since a capacity of more than 256 records is desired TWO bytes (lo and hi) are used in the position command to "get at" the specified record number. TWO bytes can handle $256 \times 256 = 65,636$ different numbers.² Here's how it works if you don't already know.

Let RN = record number
LO = contents of 1st byte
HI = contents of 2nd byte

The formula: $RN = 256 \times HI + LO$

² - If that is not enough records, it's time to consider a different system.

7335 (Cont'd)

Examples: If RN = 42, then LO = 42 and HI = 0

$$\begin{array}{r}
 \text{HI} \quad \text{LO} \\
 \downarrow \quad \downarrow \\
 256 * 0 + 42 \\
 = 0 + 42 \\
 = 42 \\
 \text{RN}
 \end{array}$$

If RN = 256, then LO = 0 and HI = 1

$$\begin{array}{r}
 256 * 1 + 0 \\
 256 + 0 \\
 256
 \end{array}$$

If RN = 700, then LO = 188 and HI = 2

$$\begin{array}{r}
 2 \leftarrow \text{HI} \\
 \hline
 256 \overline{) 700} \leftarrow \text{RN} \\
 \underline{512} \\
 188 \leftarrow \text{LO}
 \end{array}$$

If RN = 1200, then LO = 176 and HI = 4

$$\begin{array}{r}
 4 \leftarrow \text{HI} \\
 \hline
 256 \overline{) 1200} \leftarrow \text{RN} \\
 \underline{1024} \\
 176 \leftarrow \text{LO}
 \end{array}$$

Try these: (answers below)³

a) If RN = 1000, then LO = _____ and HI = _____

b) If HI = 2 and LO = 3 then RN = _____

Rem: Here we'll trace some values through the algorithm 8890 - 8891.

8890 HI = INT(RN/256) "INT" is the BASIC function that "chops" the remainder.

Suppose RN = 700

HI = INT (700/256)

HI = INT (2.734375)
chopped
 HI = 2 ← result of 8890

³ - a) LO = 232 & HI = 3; b) RN = 256*2+3=512+3=515



7336 (Cont'd)

9199 RETURN to 2011 which gives warning but allows writing
4011 which will return to menu if record no written (* was found)
or allow existing record to be read.

7337 9500 REM ** WAIT ***

This routine is encountered whenever the user needs an escape option. If you use the program you may add these as needed or delete the annoying ones. You could also delete 9540 to allow "any key" to continue (that could be dangerous).

```
9510 PRINT "MAYBE - C TO CONTINUE / + ESCAPE TO MENU"
9520 GETK$: IFK$="" THEN 9520
9530 IFK$="+" THEN PRINT "J": GOTO 340
9540 IFK$<>"C" THEN 9520
9599 RETURN to 1121 or 1140 or 12212 or 1240 or 1350 or 1550 or 2029
```

7338 9999 REM ** ERROR CHANNEL READ **

```
10000 INPUT #15, EN, EM$, ET, ES
```

FILE number of command channel from 300

Error number

Error message string

Error track #

Error sector #

This reads the DOS for possible problems.

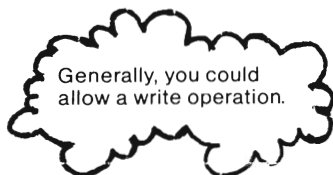
```
10010 IF EN < 20 THEN RETURN to 321 or 1791 or 2071 or 4051 or 7781 or 9131
```

The "All Clear"

```
10020 IF EN = 50 THEN IF M < 30 OR M > 7 THEN RETURN 1186, 1286, 7781 allows CREATE, EX-  
PAND, CHANGE FILE or END RUN.
```

EN = 50, RECORD NOT PRESENT (7150)

```
10030 IF EN = 50 THEN PRINT EN$, RN: GOTO 340 refuses to allow reading (probably already  
caught in 9100 "RECORD CHECK") or  
writing non-existent record.
```



² - If you choose to add it.

7338 (Cont'd)

```
10040 PRINTEN;EM$,ET;ES:CLOSE3:CLOSE15
```

Informs user of unscheduled error condition (e.g. somehow the disk got out of the drive) and protects data, especially any in a buffer, from loss.



7339 19999 END In this program END occupies a separate line to prevent secondary program from running. It would work just as well appended to 10040.

```
7340 20000 REM ***** SECONDARY PROGRAM *****
```

Rem: This program, independent of the main, was originally interded as a "debugging device." It is included in the final version to illustrate a few different techniques and to allow user to examine file contents. Results of runs are included in addition to the explanation for those beyond the casual user stage.

7341 Explanation

```
20100 PRINT"J":OPEN15,8,15
20110 INPUT"FILE NAME";FF$
```

command channel. file number chosen to match.

```
20120 OPENS3,8,3,FF$:GOSUB21000
```



Error Channel read routine for this program.

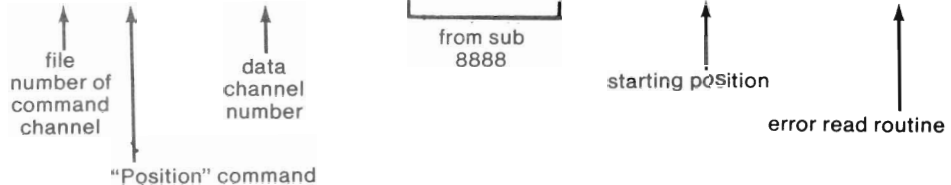
Note use of short form to "open to use" existing relative file (7142)

```
20130 INPUT"RECORD #";RN:GOSUB8888
20140 INPUT"LENGTH";L
```

"RECORD # ALGORITHM" from main program.

"Length" is somewhat of a misnomer. The number here is the number of characters to be retrieved from the file.

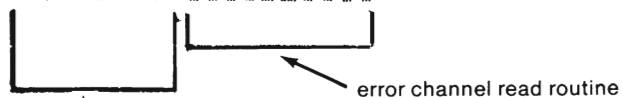
```
20150 PRINT#15,"P"CHR$(3)CHR$(LO)CHR$(HI)CHR$(1):GOSUB21000
```



```
20160 FORI=1TOL beginning of character retrieval loop
```

7341 (Cont'd)

```
20170 GET#3,G$:GOSUB21000
```



Notice use of GET# command. It will read the one character from the file at each pass through the loop. A string variable, G\$, was used to allow for reading of all characters.

```
20175 IFG$=CHR$(13)THENG$="↵"      uses techniques of (6260) to show computer's eye
20180 IFG$=""THENG$="%"             view of file
20185 PRINTG$;
20190 NEXT
```

```
20200 PRINTCHR$(13)"DOANOTHER RECORD?(Y/N)"
20210 GETK$:IFK$=""THEN20210
20220 IFK$="Y"THEN20130
20999 CLOSE3:CLOSE15:END
```

Only a "Y" response here will recycle. Any key will end. This secondary program does not protect the user as as the main program.

```
21000 INPUT#15,EN,EM$,ET,ES
21010 IFEN<20THENRETURN
21020 PRINTEN;EM$;ET;ES:CLOSE3:CLOSE15:END
```

This error read routine doesn't even check for Error #50 but simply lets the program abort when the file ends

7342 - 7344 Sample Runs

7342

```
GR 1
RN= 1 L= 1000
ONE+*****1+*****1+*****
 1+1+1+*****1+*****1+*****1+NEW 2+
 *****2+*****2+*****2+2+2
 2+*****2+*****2+JOHN+*****
 3+*****3+*****3+3+3+*****3+
 *****3+*****3+NANCY+*****4+
 *****4+*****4+4+4+*****4+
 *****4+PAT+*****5+*****5+
 *****5+5+5+*****5+*****5+
 JIM+*****6+*****6+*****6+
 *****6+6+6+*****6+*+TEST+
 *****DASH+*****-+*****-+
 *+*****+*****+*****+*****
```

Shows typical Test file, (See also (7249).)

␣ blank in file

+ carriage return CHR*(10)

← note: the * shows existing but unwritten record (# 7)

records created by job # 1 or job # 2 but not written.

↑↑ marked by system, these "records" fill left overs in data block but do not "exist" (7321)

7343

File Evolution

```
NEW FILE
RN= 1 L= 1000
****
```

shows NEW FILE. This file has been opened in 320 but no records created by job # 1 or job # 2

```
NEW FILE
RN= 1 L= 1000
*+*+*+*+*+*+*+*
```

shows NEW FILE after the creation of six records by job # 1.

```
NEW FILE
RN= 1 L= 1000
*+*+NUMBER 3+*****NEW FILE+*****333 33
 3333+3+*+9.00+3 MAR 83+333 3333+*+*+*+
 *+
```

shows NEW FILE after record # 3 was written.

7344

File Expansion

```
GR 1
RN= 1 L= 1000
ONE+*****1+*****1+*****
 1+1+1+*****1+*****1+*****1+*+*+*+NANC
 Y+*****4+*****4+*****4+
 4+4+*****4+*****4+*****4+PAT+*****
 *****5+*****5+*****5+5+5+*****
 5+*****5+*****5+JIM+*****6+
 *****6+*****6+6+6+*****6+
 *****6+*****6+*+TEST+*****DASH+*****
 *****-+*****-+*****-+*****-+
 *****-+*+*+*+*+*
```

shows Test File from (7342) after job # 2 destroyed records #2 and #3

7400 "GEN-REL" - Complete Listing

```

100 REM *** GEN REL ***
110 READD
115 REM DIM DC$(D),L(I),P(I),D$(I)
120 FORI=1TOD
125 READDC$(I),L(I)
130 P(I)=L(I-1)+P(I-1)+1
135 L=P(I)+L(I)
140 D$(I)="-"
145 NEXT
200 PRINT"FILE FORMAT"
210 PRINT"CODE"TAB(20)"LENGTH" POSITION
220 FORI=1TOD
230 PRINTDC$(I)TAB(20)L(I)TAB(30)P(I)
240 NEXT
250 PRINT"RECORD LENGTH",L
255 IFL>254THENPRINT"TOO LONG":END
260 PRINT"TYPE - C TO CONTINUE / ← ESCAPE PROGRAM"
265 GETK$:IFK$=""THEN265
270 IFK$="←"THENEND
280 IFK$="C"THEN265
300 OPEN15,8,15
310 INPUT"FILE NAME":FF$
320 OPEN3,8,3,FF$+",L,"+CHR$(L)
321 GOSUB9999
325 PRINT"ANY KEY TO CONTINUE"
327 GETK$:IFK$=""THEN327
330 PRINT"
340 PRINTTAB(18)"MENU":PRINT"FILE: "FF$
341 PRINT" 1 - CREATE NEW FILE
342 PRINT" 2 - EXPAND FILE
343 PRINT" 3 - WRITE IN SEQUENCE
344 PRINT" 4 - WRITE INDIVIDUAL RECORD
345 PRINT" 5 - READ IN SEQUENCE
346 PRINT" 6 - READ EXISTING RECORD
347 PRINT" 7 - EDIT EXISTING RECORD
348 PRINT" 8 - CHANGE FILE
349 PRINT" 9 - END RUN
350 IFM>0THENPRINT"LAST JOB: "M
355 PRINT"FILE: "FF$, REC # "RN"
360 INPUT"JOB #":M
370 IFM<1ORM>9THEN360
380 IFM=8THENCLOSE3:GOTO310
390 IFM=9THENCLOSE3:CLOSE15:END
400 ONMGO1100,1200,1300,1400,1500,1600,1700
910 DATA9
911 DATA LAST NAME,16
912 DATA FIRST NAME,12
913 DATA SOC SEC #,11
914 DATA CLASS(9-12),2
915 DATA SEX(M/F),1
916 DATA AGE(X,XX),4
917 DATA D.O.B.(XX-XX-XX),8
918 DATA PHONE#(XXX-XXXX),8
919 DATA COMMENT,9

```

(Continued)

7400 (Continued)

```

1100 REM ** CREATE NEW FILE **
1110 PRINT"CREATE:" "FF$
1120 PRINT"WILL DESTROY DATA IF USED ON OLD FILE"
1121 GOSUB9500
1130 PRINT"DIRECTORY ENTRY CREATED"
1135 PRINT"ANTICIPATED NUMBER OF RECORDS":INPUTN
1140 GOSUB9500
1160 P=1
1165 FORI=1TON
1170 RN=I
1175 GOSUB8888
1180 GOSUB7777
1185 PRINT#3,"*"
1186 GOSUB9999
1190 NEXT
1199 GOTO340
1200 REM ** EXPAND FILE **
1210 PRINT"EXPAND:" "FF$
1220 PRINT"WILL DESTROY DATA IF USED ON OLD FILE"
1230 PRINT"CURRENT LAST RECORD #":INPUTS
1235 PRINT"NUMBER OF RECORDS TO ADD":INPUTA
1240 GOSUB9500
1260 P=1
1265 FORI=S+1TOS+A
1270 RN=I
1275 GOSUB8888
1280 GOSUB7777
1285 PRINT#3,"*"
1286 GOSUB9999
1290 NEXT
1299 GOTO340
1300 REM ** WRITE IN SEQUENCE **
1305 PRINT"WRITE IN SEQUENCE STARTING WITH"
1310 INPUT"RECORD #":S
1315 INPUT"WRITE TO #":A
1320 FORJ=STOR
1330 RN=J
1340 GOSUB2000
1350 GOSUB9500
1360 NEXTJ
1399 GOTO340
1400 REM ** WRITE INDIVIDUAL RECORD **
1410 INPUT"RECORD #":RN
1420 GOSUB2000
1430 PRINT"ANY KEY FOR MENU"
1435 GETK$:IFK$=""THEN1435
1499 GOTO340
1500 REM ** READ IN SEQUENCE **
1505 PRINT"READ IN SEQUENCE STARTING WITH"
1510 INPUT"RECORD #":S
1515 INPUT"READ TO #":A
1520 FORJ=STOR
1530 RN=J
1540 GOSUB4000
1550 GOSUB9500
1560 NEXTJ
1599 PRINT"J":GOTO340

```

(Continued)

7400 (Continued)

```

1600 REM ** READ INDIVIDUAL RECORD **
1610 INPUT"RECORD #";RN
1620 GOSUB4000
1630 PRINT"ANY KEY FOR MENU"
1635 GETK$:IFK$=""THEN1635
1699 GOTO340
1700 REM ** EDIT RECORD **
1710 INPUT"RECORD #";RN
1715 GOSUB4000
1720 PRINT"ATYPE: 0 - NO CHANGE
1721 PRINT"      10 - CHANGE ALL
1722 PRINT" 1 TO 9 - FIELD CHANGE
1725 INPUTN:IFN<0ORN>10THEN1725
1730 IFN=0THEN340
1735 IFN=10THENGOSUB2030:GOTO1715
1740 PRINTD$(N)
1745 INPUTI$(N)
1750 D$(N)=LEFT$(D$(N),L(N))
1755 PRINT"ANOTHER FIELD?(0-9)":INPUTH
1760 IFH=0THEN1775
1765 IFN<10RN>9THEN1755
1770 GOTO1740
1775 FORI=1TOD
1780 P=P(I)
1785 GOSUB7777
1790 PRINT#3,D$(I)
1791 GOSUB9999
1795 NEXT
1799 GOTO1715
1999 REM **** SUBROUTINES ****
2000 REM ** WRITE RECORD **
2005 PRINT"RECORD #";RN
2010 GOSUB8888
2011 GOSUB9100
2020 IFC$="*"THEN2029
2025 PRINTRN" HAS BEEN WRITTEN"
2029 GOSUB9500
2030 FORI=1TOD
2035 P=P(I)
2040 PRINTD$(I) ["L(I) SPACES"]
2050 INPUTI$(I)
2055 D$(I)=LEFT$(D$(I),L(I))
2060 GOSUB7777
2070 PRINT#3,D$(I)
2071 GOSUB9999
2080 NEXT
2099 RETURN
4000 REM ** READ RECORD **
4010 GOSUB8888
4011 GOSUB9100
4020 IFC$="*"THENPRINT"RECORD #";RN" NOT WRITTEN":GOTO340
4025 PRINT"FF$TAB(20)"#";RN"

```

(Continued)

7400 (Continued)

```

4030 FORI=1TOD
4035 P=P(I)
4040 GOSUB7777
4050 INPUT#3,D$(I)
4051 GOSUB9999
4060 PRINTI" "DC$(I)TAB(22)D$(I)
4070 NEXT
4099 RETURN
7777 REM ** POSITION POINTER **
7780 PRINT#15,"P"CHR$(3)CHR$(LO)CHR$(HI)CHR$(P)
7781 GOSUB9999
7799 RETURN
8888 REM ** RECORD # ALGORITHM **
8890 HI=INT(RN/256)
8891 LO=RN-HI*256
8899 RETURN
9100 REM ** CHECK RECORD **
9110 P=1
9120 GOSUB7777
9130 GET#3,C$
9131 GOSUB9999
9140 IFC$=CHR$(255)THENPRINT"☐RN" NOT PRESENT":GOTO340
9199 RETURN
9500 REM ** WAIT ***
9510 PRINT"TYPE - C TO CONTINUE / + ESCAPE TO MENU"
9520 GETK$:IFK$=""THEN9520
9530 IFK$="+ "THENPRINT"☐":GOTO340
9540 IFK$<>"C"THEN9520
9599 RETURN
9999 REM ** ERROR CHANNEL READ **
10000 INPUT#15,EN,EM$,ET,ES
10010 IFEN<20THENRETURN
10020 IFEN=50THENIFM<30RM>7THENRETURN
10030 IFEN=50THENPRINTEM$,RN:GOTO340
10040 PRINTEN;EM$,ET;ES:CLOSE3:CLOSE15
19999 END
20000 REM **** SECONDARY PROGRAM ****
20100 PRINT"☐":OPEN15,8,15
20110 INPUT"FILE NAME";FF$
20120 OPEN3,8,3,FF$:GOSUB21000
20130 INPUT"RECORD #";RN:GOSUB8888
20140 INPUT"LENGTH";L
20150 PRINT#15,"P"CHR$(3)CHR$(LO)CHR$(HI)CHR$(1):GOSUB21000
20160 FORI=1TOL
20170 GET#3,G$:GOSUB21000
20175 IFG$=CHR$(13)THENG$="+ "
20180 IFG$=""THENG$="%"
20185 PRINTG$;
20190 NEXT
20200 PRINTCHR$(13)"☐ANOTHER RECORD?(Y/N)"
20210 GETK$:IFK$=""THEN20210
20220 IFK$="Y"THEN20130
20999 CLOSE3:CLOSE15:END
21000 INPUT#15,EN,EM$,ET,ES
21010 IFEN<20THENRETURN
21020 PRINTEN;EM$,ET;ES:CLOSE3:CLOSE15:END

```



8000 RANDOM FILES**CHAPTER DIRECTORY**

8100	RANDOM Files in General
8110	Overview
8120	Command format & function
8200	Sample Program
8210	Program Objectives
8230	New Instructions
8300	Explanation
8310	Initial Set Up
8320	Writing Random Files
8330	Reading Random Files
8340	"BLOCK ALLOCATE"
8350	Examine Sequential File
8360	Error Channel Read
8400	LISTING of "RANDOM FILES"
8500	An Afterthought

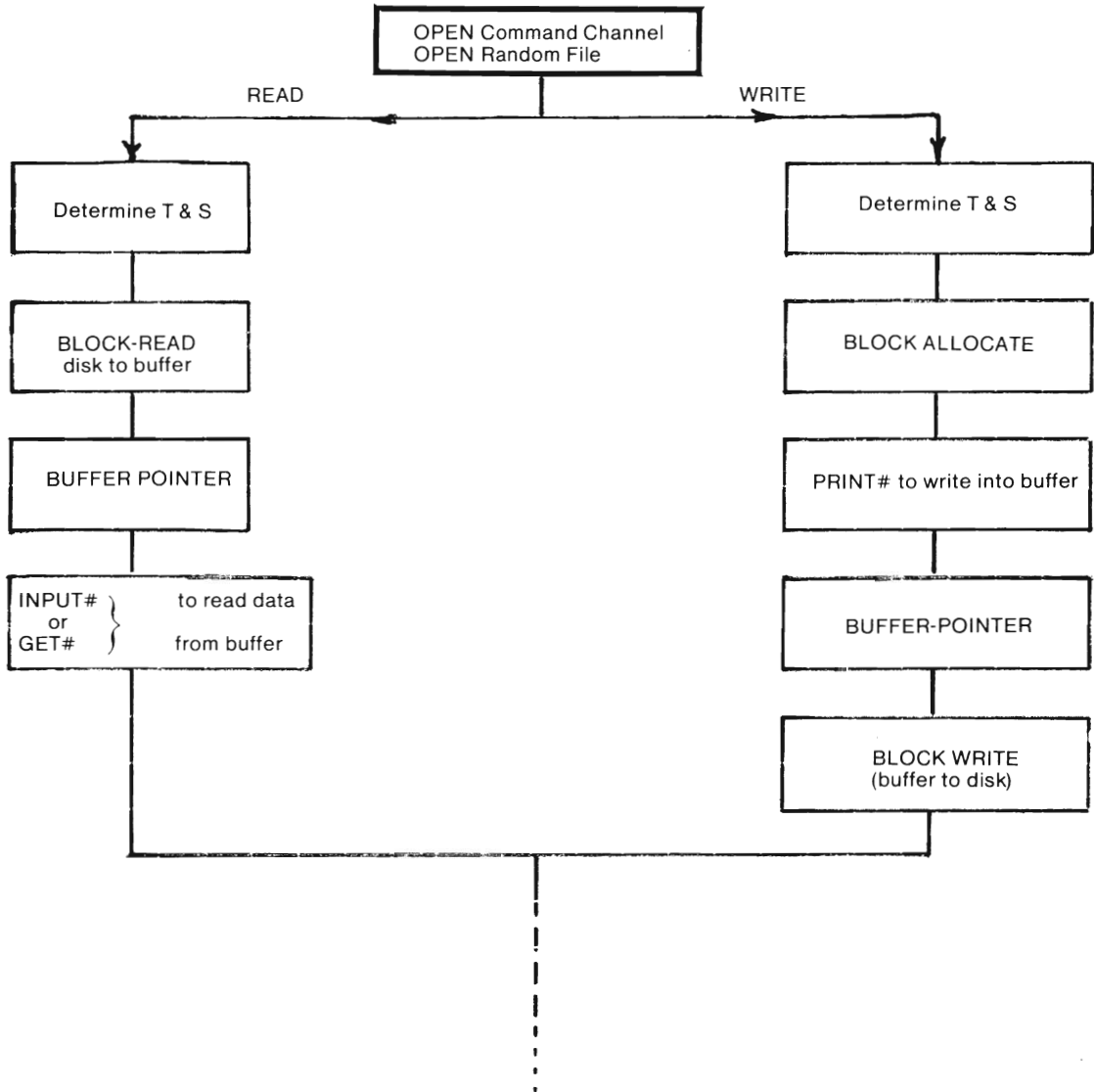
8100 RANDOM Files in General

Rem: Although RANDOM files can be used for the same purposes as RELATIVE files, they are more awkward and require a great deal more bookkeeping. One forte of RANDOM files is utility routines such as those found on your TEST/DEMO DISK.²

This chapter, I'm sorry to say, doesn't really get into matters as deeply as the preceding nor is it as detailed.

8110 Overview

Rem: RANDOM file processes allow access to any block (on a formatted disk) given to track and sector numbers. A careful examination of (0260) and (0270) may be in order.



² - © Commodore Business Machines.

8120 **Command Format & Function**8121 **OPEN**format: `OPENfn,dv,ch,"#"`parameters: *fn* = file number
dv = device number
ch = channel number

"#" is the symbol used to indicate a RANDOM file as opposed to a sequential or relative. RANDOM files do not have "names" as such. "#" is always the "name" of the random file.

ex: OPEN10,8,10,"#"
OPEN10,8,84,"#"8122 **BLOCK-ALLOCATE**format: `PRINT#15,"B-A:"0;t;s`

parameter: 0 = drive number (0150)

t = Track number - numeric variable*s* = Sector number - numeric variable

Assuming command channel opened as OPEN15,8,15 .
If a different file number were chosen, all 15's in these formats would, of course, be changed to that number.

"B-A:" the command string. The words "BLOCK-ALLOCATE" may be used if desired.

function: This command, along with a check on the error (command) channel, checks BAM(0270) to see if the specified block is available. If so, the program allows the write operation. If not, DOS sends "65 - NO BLOCK" and sets Track and Sector numbers to those of the next one available. In either case, a block is allocated those of BAM.

e.x. `PRINT#15,"B-A:"0;10;2` ← Sector 2

Track 10
↓

checks to see if track 10 sector 2 is available. If available, (10,2) is then marked. If not available a reading of the error channel will indicate the next available block by Track and Sector numbers and it is marked as used

8123 **BLOCK-FREE**format: `PRINT#15,"B-F:"0;t;s`

parameters: as in (8122)

"B-F:" is the command string. The words "BLOCK - FREE" may also be used.

Function: This command causes BAM to mark a block as available. When a block is available it can be written on by any write command. In some ways it is similar to the SCRATCH command (4500). The VALIDATE command (4400) "frees" all allocated blocks that have not been written whereas BLOCK-FREE "frees" them one at a time.

Notice that the last few punctuation marks are semi-colons.²

² - Some of these commands work with commas in place of the semi colons. Some don't like commas. Maybe you can experiment and figure out why.

8124 **BLOCK-READ**format: `PRINT#15,"B-R:"ch;0;t;s`parameters: *ch* = channel number from random file data channel. Thus as in (8122).function: This command feeds the contents of the block specified by T & S into the data channel buffer until the buffer pointer signals the end of data. INPUT # *fn*/or GET# *fn* are then used in the usual manner to read the data. Use of INPUT# requires the usual cautions with file data format. (5400)8125 **BLOCK-WRITE**format: `PRINT#15,"B-W:"ch;0;t;s`

parameters: as in (8124)

function: After information is placed into the data channel buffer via PRINT#*fn*, this command does the real work. The DOS takes care of character counting and places end of data markers for use in read operation. i.e., a buffer pointer notes the position of the last character written.8126 **BUFFER-POINTER**format: `PRINT#15,"B-P:"ch;p`parameters: *ch* = channel number of random file data channel.*p* = position within block (0 - 255)function: This command is similar to the "POSITION" command in Relative files. With relative files, we position relative to the beginning of the *record*. With random files we position from the beginning of a *data block*.

(Same note as on last page)

8127 **CLOSE**format: `CLOSE/n`parameter: *fn* = file number of random file data channel.function: as described in preceding chapters *ad nauseum*, but still as important.8128 **USER1**format: `PRINT#15,"U1:"ch;0;t;s`parameter: *ch*, *0*, *t*, *s* as in (8124)

"U1:" command string. ("UA:" is an equivalent command.)

function: This command is similar to BLOCK-READ except that it feeds all 256 bytes into the data channel buffer regardless of pointers that signal end of data.

8129 **USER2**

format: **PRINT#15,"U2:"ch;0;t;s**

parameter: (8128)

"U2:" command string ("UB:" is an equivalent command.)

function: This command is similar to BLOCK-WRITE except that it does not alter the buffer printer that contains the position of the last character written.

8200 SAMPLE PROGRAM

8210 PROGRAM OBJECTIVES

8211 This program places a user's string into a Track and Sector selected by the DOS and keeps track of these via a sequential file.

8212 The primary intent of this program is that of a learning experience rather than that of a user oriented program. It is also an example of a grade D program.

8220 USER INSTRUCTIONS

8221 Preliminaries
PROGRAM NAME: "RANDOM FILES"
(normal load)

Read RND SAMPLE first.
Do NOT write before you read (8224)

You will be asked for a file name. RND SAMPLE exists on "Friendly Floppy" so use that for your first attempt. Otherwise the program will try to read or write the sequential file you name. If your intent is to write your own file feel free to use the name of your choice so long as it does *not* exist on the disk in the drive. (8224)

8222 READING

After you've given the program a file name, say RND SAMPLE, the program asks if you wish to read or write. Say READ for now. All you'll need is R.

Now it wants to know which record to read. If you're reading RND SAMPLE ask for record #2 and you see something like this.

	N		T		S		ST	
	1		1		1		0	} from Program line 440 (8330)
	2		1		2		0	
	# 2	TRACK-	1	SECTOR	2			← from Program line 500 (8330)

#2
THUMB

from Random file. (all else from sequential file)
program line 530 (8330)

8222 READING (Cont'd)

The Random file contains only the number, and a string, 2 and Thumb in the case of RND SAMPLE. The other things you see will be explained later (8300).

When the program asks for another Type: Y followed by # 7 and you'll see something like this.

N		T		S		ST	
1		1		1		0	} from sequential file named
2		1		2		0	
3		1		3		0	
4		1		4		0	
5		1		5		0	
6		1		6		0	
7		1		7		64	

7 TRACK- 1 SECTOR 7

#7
WIDGET

from the Random file, (8330), program line 530

Fool around with reading other records in this file. Try # 10 for instance to see what happens.

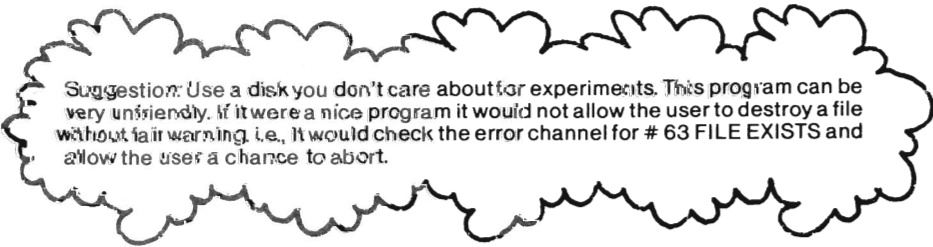
8223 The Secondary Program

Type: N for no more records then
Type: RUN 9000 to run the secondary program. You'll see something like this.

			record #	
			track #	
			sector #	
1	,	1	,	1
2	,	1	,	2
3	,	1	,	3
4	,	1	,	4
5	,	1	,	5
6	,	1	,	6
7	,	1	,	7

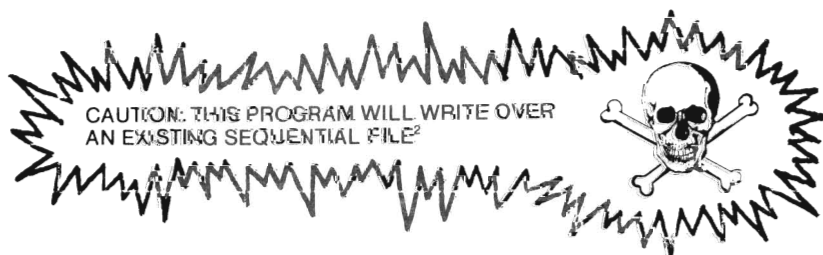
These are the contents of the companion sequential file. With the program as written, RND SAMPLE will always be read. To change to your file LIST 9000 and change the name. Better yet, use 9000 INPUT "FILE";FF\$ 9005 OPEN 5,8,5,FF\$+" ,S,R"

8224 WRITING



Suggestion: Use a disk you don't care about for experiments. This program can be very unfriendly. If it were a nice program it would not allow the user to destroy a file without fair warning, i.e., It would check the error channel for # 63 FILE EXISTS and allow the user a chance to abort.

When you run the program, supply a file name that is *not* on the disk. This will be the name of a sequential file that will contain the record number, the track and sector numbers of that record.



Next you'll be asked for a string. You may use anything you want. You'll hear some buzzing and then see the track and sector numbers of the block on which your data was placed. Hit N when you get sick of writing records. Run again and read them using the proper name for the file of course. If you want to examine the contents of your sequential see the explanation section (8223)

8300 EXPLANATION

Rem: This program is far from being slick and user friendly. It is intended merely as a sample of some of the commands used for random files. As usual, program line references are not in "()" while references to text lines are in "()".

8310 Initial Set Up



```
10 PRINT"RANDOM FILES"
15 INPUT"FILE NAME";FF$:PRINT"READ OR WRITE"
20 GET$:IFK$=""THEN20
21 IFK$="W"THENGOSUB100
22 IFK$="R"THENGOSUB400
23 END
```

Line 15 can get the user into big trouble. There should be a "quit" option included in the so called menu. Also in line 15, line 23 will simply stop the program if the user does not select R or W.

RUN/STOP and **RESTORE** is another way out.

² - The replace option (6150) is in effect. It will replace the existing file and has been known to mess up other directory entries.

8320 Writing Random Files

```

99 REM **** TO WRITE RANDOM FILE ****
100 OPEN15,8,15,"I":M#=CHR$(44)
110 OPEN10,8,10,"#"
120 OPEN5,8,5,"@:"+FF#+",S,W":N=1:GOSUB9900

```

100 opens the command channel and initializes (4300) the disk in the process. CHR\$(44) is just a comma for data input. (5400)

110 opens the random file. (8121)

120 opens the companion sequential file with the replace option in effect. You may wish to remove it. (6150)

The first record we write will be record # 1. 9900 is the usual "error channel" read routine.

```

200 N#=STR$(N):INPUT"DATA $":D#

```

Here the record number is converted to a string and the user asked for the data. A great deal more data could be placed in this record since an entire block will be set aside for this entry. Using an entire block for a single string is a waste.

```

210 PRINT#10,N#M#D#
220 GOSUB8000

```

210 sends information to the buffer

220 sends the system to the "Block Allocate" routine. (8340)

```

230 PRINT"TRACK-"T,"SECTOR-"S
240 PRINT#5,N;M#;T;M#;S

```

M\$ is just "

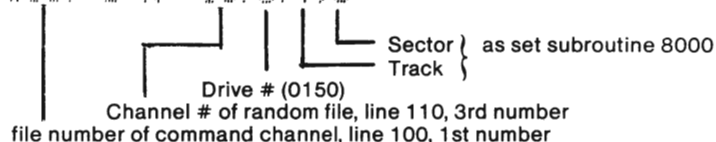
230 simply informs the user of the track and sector numbers selected by the DOS. In practice the user does not need this information since it is contained in the companion sequential file. This line 230 could be deleted without affecting the files in any way.

240 writes information to the sequential file. Note the use of semicolons to separate numerix from string variables. As you can see the sequential file contains the record number along with its assigned track and sector.

```

250 PRINT#15,"B-W:"10;0;T;S

```



This command causes the information in the buffer, from line 210, to be placed on track T, sector S.

```

260 PRINT"ANOTHER(Y/N)
261 GETK$:IFK#=""THEN261
262 IFK#="Y"THENN=N+1:GOTO200
263 IFK#<>"N"THEN261
270 CLOSE5:CLOSE10:CLOSE15
299 RETURN

```

Here the user has the option to write another record. Line 262 increases the record # by one automatically.

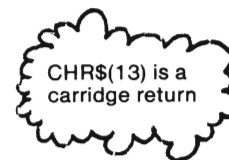
8320 Writing Random Files (Cont'd)

Note: If we were using the "Buffer Pointer" command (8126) to separate this block into fields, we would use that command prior to line 210.

Line 270 closes all files, sequential, random and command respectively. In practice only the sequential needs to be closed when changing from read to write operations. The random file and the command channel could be opened at the beginning of the program and closed at the end as in "GEN REL" (7000)

8330 READING RANDOM FILES

```
399 REM ***** TO READ RANDOM FILE *****
400 OPEN#5,8,15,"I":C#=CHR$(13)
410 OPEN#6,8,10,"#"
420 OPEN#5,8,5,FF#+",S,R":GOSUB9900
```



The lines 399, 400 and 410 are nearly identical to lines 99, 100 and 110 respectively. Line 420 opens the sequential file to read.

```
430 PRINT"WHICH RECORD TO READ?"
   :INPUTN:PRINTC#" #"," T"," S", "■■■■STATUS" C#
```

Most of this line consists of screen headings in addition to asking the user for the record number. In practice there is no need for the last batch of headings. They are there to help you explore the STATUS function as it relates to some disk operations.

```
440 INPUT#5,N,T,S:PRINTN,T,S,ST
```

Record number, Track and Sector numbers are received from the sequential file (fn=5) and printed to the screen.

ST is the STATUS of the input/output device. This print statement is for information purposes only. Now would be a good time to take a look at the numbers in the tables in (8222)

```
450 IFST=64THENPRINT"LAST RECORD IN FILE"
```

Actually, STATUS of 64 is all we'd expect to find. 64 signals the end of the file. The sequential file programs in (6000) could have used the status function instead of the flag system. "COMMAND DEMO" (6300) uses the STATUS variable to "find the end" of the file. Line 450 serves only as an aid to file exploration.

```
460 IFKORNFANDST=64THENGOTO440
465 IFN=RNTHEN500
```

460 causes the system to keep reading the sequential file for the desired record number. The actual read process for the random file won't start until we reach 500.

```
470 PRINT"RECORD NOT FOUND":GOTO540
```

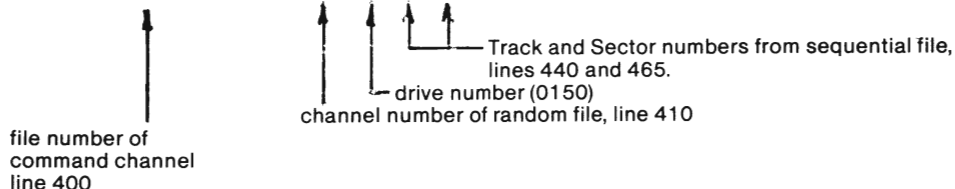
This line is simply user information and is not executed if the record is found.

8330 READING RANDOM FILES (Cont'd)

```
500 PRINT"#N" TRACK="T" SECTOR"S
```

This line displays information from the sequential file when the record has been found. It is not necessary for random file operations. Information from this line should, of course, agree with that from line 230 when this record was written.

```
510 PRINT#15,"B-R:"10;0;T;S
```



This "BLOCK-READ" command sends information from the disk into a buffer.

```
520 INPUT#10,N$,D$
```

N\$ = STR\$(N), the record number

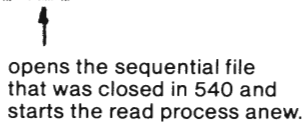
Receives information from the RANDOM file via the buffer.

```
530 PRINTC$"#N$C$D$" simply displays the information received in 520
```

```
540 CLOSE5:PRINT"READ ANOTHER(Y/N)"
```

The sequential file is closed at this time so that it can be reread from the beginning. Without the close, if the users last request was for record # 10, the next reading would start with record # 11. This is fine if the users next request were for record # 15 but would result in "record not found" if the user had requested any record less than # 11.

```
541 GETK$:IFK$=""THEN541
542 IFK$="N"THENCLOSE10:CLOSE15:RETURN
543 IFK$<>"Y"THEN541
550 GOTO420
```



to line 22 and then to the end of the program. This is another of the "not so nice" parts of the program

8340 "BLOCK ALLOCATE"

```

8000 REM*****BLOCK ALLOCATE*****
8010 T=1:S=1
8020 PRINT#15,"E-A:"0,T,S
8030 INPUT#15,EN,EM$,ET,ES
8040 IFEN=65THENET=ET:S=ES:GOTO8020
8050 RETURN
8900 END

```

Line 8010 starts the search for a free block at Track - 1, Sector - 1. Strictly speaking, such is not necessary. We could start the search at any Track and Sector number. Now would be a good time to re-examine the TRACK - SECTR TABLE in (0233)

8020 causes the DOS to examine the BAM (0270) for Track 1, Sector 1. If free that block will be allocated. If not, the DOS contains a message, EN (error number) = 65 and EM\$ (error message) = "NO BLOCK"

8030 reads that message. ET (error track) and ES (error sector) are the numbers of the next available block on the disk.

8040 sets the new Track and Sector numbers and returns to 8020 to allocate that block.

8050 returns to line 220 to continue the write process.

8900 is totally unnecessary and should have been removed.

8350 Examining the Sequential file

```

8998 REM*****GET SEQ FILE*****
8999 REM**      RUN 9000      **
9000 OPENS,8,5,"RND SAMPLE,S,R"
9010 GET#5,G$:PRINTG$;
9020 IFST=64THENCLOSE5:END
9030 GOTO9010

```

This routine is very much like that in (6000) as explained in (6235).

Line 9020 stops the process when the disk status is 64. That's the signal for the "end of the file."

8360 Error Channel Read Routines

```

9900 REM***READ ERROR CHANNEL***
9910 INPUT#15,EN,EM$,ET,ES
9920 IFEN<20THENRETURN
9925 IFEN<20THENRETURN
9930 PRINTEN;EM$,ET,ES:CLOSE5:CLOSE10:CLOSE15:END

```

Standard routine as described earlier. You're not crazy, lines 9920 and 9925 are identical. One of them should have been removed.

8400 LISTING of "RANDOM FILES"

```

10 PRINT"RANDOM FILES"
15 INPUT"FILE NAME";FF$:PRINT"READ OR WRITE"
20 GETK$:IFK$=""THEN20
21 IFK$="W"THENGOSUB100
22 IFK$="R"THENGOSUB400
23 END
99 REM **** TO WRITE RANDOM FILE ****
100 OPEN15,8,15,"I":M$=CHR$(44)
110 OPEN10,8,10,"#"
120 OPEN5,8,5,"@":"+FF$+",S,W":N=1:GOSUB9900
200 N$=STR$(N):INPUT"DATA $";D$
210 PRINT#10,N$M$D$
220 GOSUB8000
230 PRINT"TRACK-"T,"SECTOR-"S
240 PRINT#5,N;M$;T;M$;S
250 PRINT#15,"B-W:"10;0;T;S
260 PRINT"ANOTHER(Y/N)
261 GETK$:IFK$=""THEN261
262 IFK$="Y"THENN=N+1:GOTO200
263 IFK$<>"N"THEN261
270 CLOSE5:CLOSE10:CLOSE15
299 RETURN
399 REM **** TO READ RANDOM FILE ****
400 OPEN15,8,15,"I":C$=CHR$(13)
410 OPEN10,8,10,"#"
420 OPEN5,8,5,FF$+",S,R":GOSUB9900
430 PRINT"WHICH RECORD TO READ?":INPUTRN:PRINTC$ " #", " T", " S");"STATUS"C$
440 INPUT#5,N,T,S:PRINTN,T,S
450 IFST<=0THENPRINT"LAST RECORD IN FILE"
460 IFN<=0ANDST=0THENGOTO440
465 IFN=RNTHEN500
470 PRINT"RECORD NOT FOUND":GOTO540
500 PRINT#"N" TRACK-"T" SECTOR"S
510 PRINT#15,"B-R:"10;0;T;S
520 INPUT#10,N$,D$
530 PRINTC$#"N$C$D$
540 CLOSE5:PRINT"READ ANOTHER(Y/N)"
541 GETK$:IFK$=""THEN541
542 IFK$="N"THENCLOSE10:CLOSE15:RETURN
543 IFK$<>"Y"THEN541
550 GOTO420
8000 REM*****BLOCK ALLOCATE*****
8010 T=1:S=1
8020 PRINT#15,"B-A:"0,T,S
8030 INPUT#15,EN,EM$,ET,ES
8040 IFEN=65THENT=ET:S=ES:GOTO8020
8050 RETURN
8900 END
8998 REM*****GET SEQ FILE*****
8999 REM**      RUN 9000      **
9000 OPEN5,8,5,"RND SAMPLE,S,R"
9010 GET#5,G$:PRINTG$;
9020 IFST=64THENCLOSE5:END
9030 GOTO9010
9908 REM***READ ERROR CHANNEL***
9910 INPUT#15,EN,EM$,ET,ES
9920 IFEN<20THENRETURN
9925 IFEN<20THENRETURN
9930 PRINTEN;EM$,ET,ES:CLOSE5:CLOSE10:CLOSE15:END

```


8500

An After thought

Rem: With this program, you can examine the contents of any block on the disk in the drive. The system may pick up some bytes that when printed to the screen will cause wierd things to happen. Taken out of context, and if you want to have a little fun, go ahead and enter and run it. (It is not on "FRIENDLY FLOPPY").

```

5 REM *** EX RANDOM ***
10 OPEN15,8,15
20 OPEN10,8,10,"#"
30 INPUT "T,S";T,S
40 PRINT#15,"U1:"10;0;T;S
50 GET#10,A#:PRINTA#;
60 IFST=0THEN50
70 CLOSE10:CLOSE15:END

```

Line 40 shows the use of "USER1". After you've run the program as is, change "U1" in line 40 to "B-A" and see if you can find a difference. Line 60 is for use with the "B-R" command.

User instructions. Simply give it a Track and Sector number. There is no "error channel read" routine so a flashing red light may mean you've given it an illegal track and/or sector number. You can have fun with this if your wedge is active. Refer to (233) for legal Track and Sector numbers.

To eliminate some of the strange "characters" you might try this:

Change Line 50 to 50 GET#10,A#

and add these lines. 51 A#=A#+CHR\$(0)
 52 A=ASC(A#)
 53 IFA<320RAD>127THENPRINT" A" ;GOTO60
 54 PRINTCHR\$(A#)



The Trouble with computers is that just when you think you know something they let you know that there's still a whole bunch more you don't.



9000 FLASHING RED LIGHT

CHAPTER DIRECTORY

9100	«WEDGE» Active
9200	PROGRAM Method
9210	Advantages
9220	Disadvantages
9230	Explanation
9240	How to Use
9250	Output
9300	IMMEDIATE MODE
9400	PANIC ABORT
9410	Red light flashing
9420	Red light solid
9500	CAUSES & REMEDIES
9600	Problems, problems
9610	Disk Stuck
9620	«Wedge» won't work
9630	Noises
9640	Load Errors


**IN CASE OF EMERGENCY
DON'T PANIC**

Relax and let it flash
9001 to 9005 until
something works

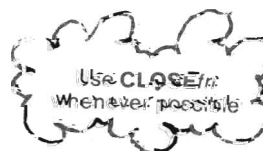


9001 Type: `IB`

9002 Check spelling, change and repeat command.

9003 Type: `OPEN15,8,15,"I"`

9004 Type: `OPEN15,8,15,"U1"`



9005 Type: (Program below. Adjust line numbers if necessary.)

```
60010 OPEN15,8,15
60020 INPUT#15,E,M#,T,S
60030 PRINT#M#,T,S
60040 CLOSE15
```

Type: `RUN60010`

9600 Preliminaries

Believe it or not, the flashing red light is your friend. Like most good friends, it can be both a big help and a real pest.

The drive flashes its red light to signal that it can't do what was asked. (The sky is probably not falling.)

Sometimes you don't even have to bother shutting it off. If you know the source of the problem and fix it, the red light will go off by itself.

Avoid removing disk when red light flashing.

9100 «WEDGE» DOS 5.1

REM: If you look at lines 9001 to 9005, it is obvious that the WEDGE ¹ command is the shortest. More than that, it is equivalent to the program process shown in 9005.


9101 Type:  ²

9102 If the red light is still flashing and all you see is "Syntax Error" your WEDGE is not active. (See 1230 for help on activating your WEDGE.) Use a different method to shut off the red light.

9103 Now you see something like this. Refer to 9500 or your user's manual to analyze error and repair.

XX, ERROR MESSAGE, XX, XX

9104 If you've been playing with files and your red light is on solid and drive is quiet,

Type:  I

(4300)



9200 THE PROGRAM METHOD

9210 Advantages of the Program Method

9211 The program presented (or variations of it) is a valuable technique when you start to write your own programs that use data files. This program "outputs" a verbal clue to the source of your error.

9212 Unless your WEDGE is active the only way to read the error messages in your DOS is via a program. Like the ordinary INPUT (5340) statement, INPUT# works only in program mode.

9220 Disadvantages of Program Method

9221 You may not enjoy typing while the red light is flashing. ← TRIVIAL

9222 If you're not careful using line numbers, you can mess up your program.

↖ CAUTION!!!


9230 Program Explained*

```
60000 REM***READ COMMAND/ERROR CHANNEL
```

↑
Line numbers are started high so that the program can be safely added to an existing program. Make necessary adjustments if your

(Continued)

¹ Don't feel sorry for yourself about having to load the WEDGE. Some systems, Brand X, require loading their entire DOSes before they can even load a program.

² > is equivalent to  and can replace it in all WEDGE commands.

* This is an elementary "how to" treatment. For generalizations see 5030.

9230 (Cont'd) program already has a line 60000. Lower line numbers may of course be used.
 You may wish to preface this with an END statement.
 ex. 59999 END

9231 60010 OPEN15,8,15

must be 15 to open command channel
 8 for unaltered drive
 FILE NUMBER. May be any number from 1 through 255, but use of 128-255 is not recommended.

NOTE: The FILE NUMBER is the one used in the following statements.

9232 60020 INPUT#15,E,M\$,T,S

May be any BASIC variables.
 The 2nd must be a string.
 The others may be string.
 Adjust next line accordingly.

E = error number in your manual
 M\$ = error message
 T = track number
 S = sector number

Use the FILE NUMBER (1st number following "OPEN" in the preceding line.)

INPUT# is the command that directs the computer to receive its information from a peripheral device. (5034)

9233 60030 PRINT#M\$,T,S

REM: This is just an ordinary PRINT statement. It directs the computer to show you what it just read from the drive. You can adjust it to any legal print format you like. You must, of course, use the same variables you used in the line above.

9234 60040 CLOSE15

FILE NUMBER. Use the 1st number from the open statement.
 CLOSSES file opened in 60010

This command will CLOSE all other files on the drive. Unfortunately your BASIC program (if you're running one) will not know it and try to read or write to a closed file. That will cause a syntax error. When programming,

OPEN command channel FIRST.
 CLOSE command channel LAST.

See also 5020 and 5032.

9240 How to Use

- 9241 In general- Type program (see below)
- Run program (see below)
- Red light goes out*
- Error message appears*
- Interpret results (9500 and user's manual)
- Repair

9242 If no program in memory, type the program with smaller line numbers. To run, just type: RUN. GO TO 9500.

9243 If you have a program in memory that you can't afford to lose, one of the following should work.

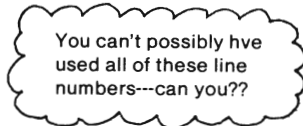
Find at least one unused line in your program. After the end of your program is most convenient. The beginning is the next best place.

At the end:

```
63999 OPEN15,8,15:INPUT#15,E,M$,T,S:PRINTE;M$,T,S:CLOSE15
```

↑ Largest allowed line number

To run, Use: RUN63999



At the beginning:

```
1 OPEN15,8,15:INPUT#15,E,M$,T,S:PRINTE;M$,T,S:CLOSE15:END
```

↑ You can even use 0.

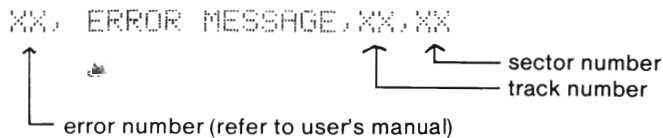
To run, use: RUN . Now you'll have to type RUN XXX to run your main program. Ugh!

9244 If you want to use this sequence of commands as part of a "file using" program, use SUB-ROUTINES. The best way to learn how is to study examples. Sequential Files (6236) and Relative Files (7338)

9250 Program Output

REM: The following is produced only by the program method or equivalentents each with active WEDGE.

9251 General Form: (controlled, of course, by the PRINT statement in the program.9233)



*If your drive is not on, nothing will happen and I mean nothing. Go on, ty it. Use RUN/STOP key and RESTORE key to get your computer back.

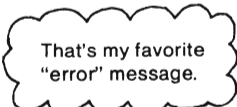
9252 Error Messages from the DOS are like those from BASIC in that sometimes they are helpful and at other times cryptic. Anything is better than nothing (maybe).

Some error messages cover several different problems. Conversely a given problem will sometimes cause different messages. Some "error" messages aren't even indicative of a problem. (#50 "record not present" in relative file usage (7152) or #1 "files scratched" in scratch operations (4500))

9253 The track and sector can also be both useful and cryptic. In simple operations you'll see mostly 0's since track and sector numbers don't apply. Generally track 18 errors refer to problems in the Directory (260) and BAM (270).

9254 If you type @ with an ACTIVE WEDGE or run the program, the red light is not flashing, you'll see.

```
00, OK,00,00
```



That's my favorite
"error" message.

9255 Section 9500 contains a list of common problems and remedies. If it doesn't help, see your user's manual.

9300 IMMEDIATE MODE COMMANDS

REM: Advantage: Easier to type.
Disadvantage: No error message

These shut off the light but do not tell you why it was on. They are great if you already know what happened and forgot to activate the WEDGE.

9301 If you get a "file open" message, it's safest to *type* CLOSE15 and then start over. (You can also start with the second line of the sequences on the right if you know that OPEN 15, 8, 15 was used earlier. If file number 15 has been opened to another device or even another drive channel, these sequences will not work until you CLOSE 15.

9302 If there is no disk in your drive, that's the problem.*

```
Type: OPEN15,8,15,"UI" or type: OPEN15,8,15
      CLOSE15                    PRINT#15,"UI"
                                   CLOSE15
```

REM: This command jumps to a special location in your drive's memory. Your drive will think it just woke up. Further discussion is beyond the scope of this text.

* You could simply insert a floppy and give another command. This practice is not recommended.

9303 If there is a disk in the drive, especially if you are working on data files,

Type: OPEN15,8,15,"I"
CLOSE15

or type: OPEN15,8,15
PRINT#15,"I"
CLOSE15

REM: "I" is short for "INITIALIZE".



9304 The "Command Sending" techniques used above are described in 4100. You'll find "INITIALIZE" discussed in 4300.

9400 **PANIC ABORT**

9410 **DRIVE QUIET/RED LIGHT FLASHING**

Simply "error condition" (0140). Your command was accurately given. You want out of the situation. You don't want to bother with preceding sections. Your «Wedge» is not active. (If it is, @ will do it).

It's up to you . . .

The risk to your software is negligible but present.

Simply: If you need to, insert or remove disk. Re-issue command.

The technique above, although not recommended, is effective.



9420 **DRIVE WORKING/RED LIGHT ON SOLID** (only occasional flashes)
Working (0140)

You have already tried RUN/STOP & RUN/STOP with RESTORE but the system won't listen.

It's up to you . . .

Both of these risk software slightly, but they may be your only alternative.

Either Turn Off the computer or Open the door to the drive.

Either will quiet the drive.

If the drive doesn't "quiet" within 15 seconds or so, shut it down. That's the real risk to the disk inside.

Remove the disk. Power up system as necessary. Cross your fingers and start over. Check out (9600)



(Continued)

9420 (Cont'd)

NOTE: If the methods of the preceding sections failed, you may have a hardware problem. See your dealer or call a consultant.

9500 CAUSES AND REMEDIES

REM: This list is not exhaustive, but does hit most common problems encountered by beginners and old timers in a hurry. Data file error messages are covered in those chapters.

NOTE: Shut off FLASHING RED LIGHT before inserting or removing a disk.

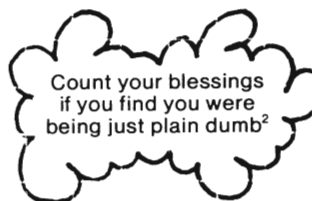
OPERATION	CAUSE	REMEDY	FOR HELP	# MESSAGE
ALL	No Disk In Drive	Insert Disk	500	74 Drive Not Ready 21 Read Error 20 Read Error
ALL	Disk Misaligned	Remove & Reinsert	600 500	74 Drive Not Ready 21 Read Error 20 Read Error
ALL (except 'format')	BLANK Disk in Drive	'format' disk (NEW)	2,000	74 Drive Not Ready 21 Read Error 20 Read Error
ALL	Improper Command	Return or Re-study	Specific Section	30 & 39 Syntax Error
LOAD	Mistyped Name	Retype	1000	62 File Not Found
LOAD	Program Not On Disk	Change Disk or Name	1000	62 File Not Found
LOAD	Attempting to Load a Data File	NONE Only PRG's & \$ Can Be Loaded	1000	64 File Type Mismatch
SAVE	Name in Use	Change Name on Disk	3120	63 File Exists
SAVE or any Write Operation	No Room On Disk or in Directory	Change Disk or Scratch Unwanted Entries	3120 4500	72 Disk Full
SAVE with Replace	Mistyped Name	Retype	3230	62 File Not Found
SAVE with Replace	Program Not On Disk	Change Disk or Ordinary Save	3120 3200	62 File Not Found
Using File Programs & Search Tricks	Changing Disks	INITIALIZE Disk	4300	29 Disk ID Mismatch 71 Directory Error
SCRATCH	Reading Error Channel After Operation	NOT a Problem Hence No Remedy	4500	1 File Scratched XX 00
ALL Especially 'Write' Operation	Using disk from Another CAM System	Additional Software	103	73 DOS Mismatch
ALL	Using Disk Prepped on BRAND 'X'	Probably None	103	74 Drive Not Ready 21 Read Error 20 Read Error ???

9600 "I can't get the disk in"

You've already opened the door.
There is no disk in the drive.

DO NOT FORCE!

Exchange disk. I have one with a bludge. It doesn't want to go in, it may not want to come out. I neither want to pay a repair bill or open up door myself.



9615 "I can't get the disk out"

The door is open, of course.
There is a disk in the drive (you can feel it there)

DO NOT FORCE!

Wobble gently. A badly taped write protect notch can hang up a disk. Simply re-tape. If it's only one disk that hangs up, exchange or discard.

If it's still stuck you need more technical help than I can give you. Call your dealer.

9620 "My «Wedge» won't work."

9621 Do you have another machine language program loaded? If so they could be in conflict. It starts at 52224. Where it goes from there or where it's ending address is I have no idea.

9622 Assuming you loaded and ran it, since the last time you powered on, did you by any chance type: @Q? If so, type SYS 52224 to re-activate.

9628 If it still won't work, you may need another copy. I know I've seen an article on how to do that, short of an entire back up, but I can't find the silly thing.

9630 "My drive is making strange noises."

Check out (0130) & (0140), & (2215). If you can identify the noise, don't worry about it, it's normal.

A soft squawking may be a forerunner of overheating. Power down as soon as possible and let it cool off for a while.

If it's been on for several hours and working hard, don't worry too much.

If it's only been on a little while you've got a problem. Consult your dealer.

If you've never experienced a "head crash," (knock wood), "they" tell me you can't mistake it. You have my deepest sympathy.

Other strange noises should be checked out by a technician.

² - I have made a truly amazing discover — none of the devices in my system work unless I plug them into the wall outlet. I should get a prize for that one!

9640 "I'm getting load errors."

I hope you're here for reference only!

9641 When was the last time you "cleaned the heads?" If you're not in the habit of doing so at least once a month, start getting into the habit now!

So, you've cleaned them and you're still having problems. ... CONTINUE

9642 If it's always the same program, and only that, that could signal software trouble. I hope you have a back up copy.

Big help aren't I?

9643 If it's always the same disk, and only that disk, guess what? I hope you have a back up.

9644 No pattern? You've got a problem. I told you earlier that FLOPSEY almost died from the strain of checking out the contents of this text. She's now been semi-retired to a loader. We don't ask her to write any more².

May I suggest you find a reliable dealer? Check out your problem. Frankly, I'm nursing the one along since the repairs will be quite expensive no matter what. On the other hand, it could be something simple. Don't give up until you get a diagnosis.

² - That tears it! Now I know I've gone over the deep end. I just hope this book gets done before I burn up another one. FLOPSEY II gets her "heads cleaned" once a week whether she needs it or not!

10000 APPENDICES
CHAPTER DIRECTORY

10100	Procedure Checks Lists
10110	System Set Up
10120	Power On
10130	Insert Disk
10140	Remove Disk
10150	Power Off
10160	LOAD
10170	"Format" (NEW)
10180	SAVE
10200	COMMAND SUMMARY (DOS 2.6)
10210	Elementary Operations
10220	General Operations
10230	Sequential File
10240	Relative Files
10250	Random Files
10260	«WEDGE» DOS 5.1
10300	DUAL DRIVE Patches
10310	Program Changes
10320	«WEDGE»
10330	Duplicate
10400	FORMS
10410	Head Cleaning Log
10420	Hardware
10430	Repairs/Maintenance
10440	Purchase Record
10450	Software
10460	Disk Inventory
10500	ID Code Check List
10600	PRINT PROGRAM LISTINGS
10610	"SEQ FILE PRINT"
10620	"GEN REL PRINT"
10630	"RANDOM PRINT"
10700	SYMBOL LIST
10800	BIBLIOGRAPHY
10900	INDEX

10100 Procedure Check Lists — Summary Only. Not for 1st time use.**10110 SYSTEM SET UP (0310)**

- Stuff gathered
- Prep Drive
- Prep Printer
- Interconnect devices
- Connect to power

10120 POWER ON (0410)

- System set up
- Cartridge Inserted
- Monitor/Television ON
- Check Drive for Disk
- Printer ON
- Drive ON
- C-64 ON

10130 INSERT DISK (0510)

- System ON
- Drive "Ready"
- Open door
- Disk from storage
- Remove from envelope
- Aim
- Insert gently
- Close Door

10140 REMOVE DISK (0610)

- Drive "Ready"
- Open Door
- Catch Disk
- Place in envelope
- Place in storage

10150 POWER OFF (0710)

- Check for disk in drive (remove)
- Computer Power OFF
- Check again for disk
- Drive Power OFF
- Printer Power OFF
- Monitor/Television Power OFF
- Last check for power lights
- Cover
- Unplug the whole mess

10160 LOAD (1010)

- System Set Up
- System ON
- Drive Ready
- Disk Inserted
- Program Name Handy
- Program in memory can be sacrificed
- Type command
- Last check on program in memory
- Hit return

10170 "Format" (NEW) (2010)

- System Set Up
- System ON
- Drive Ready
- Disk Selected (Careful here!)
- Name chosen
- ID chosen
- Check Disk Again
- Type command
- Last chance to check Disk
- Hit return

10180 SAVE (3010)

- System Set Up
- System ON
- Drive ready
- Formatted Disk Selected
- Disk write protect notch uncovered
- Disk inserted
- Name chosen
- Initialize if necessary
- Type command
- Hit return
- Verify
- Make Back up copy

10200 COMMAND SUMMARY (DOS 2.6)

10210 ELEMENTARY OPERATIONS

LOAD"PROGRAM",8	normal (1000)
LOAD"PROGRAM",8,1	special memory (1260)
SAVE"0:PROGRAM",8	normal (3000)
SAVE"0:PROGRAM",8,1	special memory (3220)
VERIFY"PROGRAM",8	(3310)

all commands
for drive 0
device 8

10220 GENERAL OPERATIONS

OPENFN, DV, 15
PRINT#FN, "COMMAND" or OPENFN, DV, 15, "COMMAND"

FN = file number
DV = device number
0 = drive number

"COMMAND"

C0: NEW FILE=0: OLD FILE	COPY (4700) new name
C0: NEW FILE=0: 1ST OLD, 0: 2ND OLD, 0: 3RD OLD	COPY (4800) to append
I0	INITIALIZE (4300)
R0: NEW NAME=OLD FILE	RENAME (4600)
N0: DISK NAME, ID	NEW (2000) format disk
N0: DISK NAME	NEW (4200) clear directory
S0: NAME	SCRATCH (4500)
V0	VALIDATE (4400)
UI	reset DOS (9302)
UI+	speed to C-64 (automatic on power up)
UI-	speed to V-20

10230 SEQUENTIAL FILES (6100)

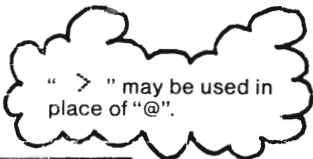
OPENFN, DU, CH, "0: FILE NAME, S, R"	open to read
OPENFN, DU, CH, "0: FILE NAME, S, W"	open to write
PRINT#FN,	write to file
INPUT#FN,	read file
GET#FN,	read byte by byte

10240 RELATIVE FILES (7100) require open command channel. Assume: OPEN15,8,15

OPENFN, DU, CH, "FILE NAME"+" ,L, "+CHR\$(RECORD LENGTH)	create
OPENFN, DU, CH, "FILE NAME"	use
PRINT#15, "P"CHR\$(CH)CHR\$(LO)CHR\$(HI)CHR\$(POSITION)	position at beginning of record
PRINT#FN,	write file
INPUT#FN,	read file
GET#FN,	read byte by byte

10250 RANDOM FILES (8100) require open command channel. Assume: OPEN15,8,15

OPENFN, DU, CH, "#"	open data file
PRINT#15, "B-A: "0; TRACK; SECTOR	Block - Allocate
PRINT#15, "B-F: "0; TRACK; SECTOR	Block - Free
PRINT#15, "B-W: "CH; 0; TRACK; SECTOR	Block - Write (buffer to disk)
PRINT#15, "B-R: "CH; 0; TRACK; SECTOR	Block - Read (disk to buffer)
PRINT#15, "B-P: "CH; POSITION	Position write beginning of block
PRINT#15, "UA: "CH; DU; TRACK; SECTOR	read entire block from buffer
PRINT#15, "UB: "CH; DU; TRACK; SECTOR	write entire buffer to block
PRINT#FN,	write to buffer
INPUT#FN,	read from buffer
GET#FN,	read from buffer byte by byte

10260 «WEDGE» Command Summary² (DOS 5.1)


" > " may be used in place of "@".

@	Read error (command) channel
@\$	View directory without overwriting memory
@Q	Deactivate «Wedge»
SYS52224	Reactivate ³ «Wedge»
/PROGRAM	normal load
%PROGRAM	special memory load
↑PROGRAM	normal load with automatic run
<0: PROGRAM	normal save*
@COMMAND	send any command (10200)
@0: COMMAND	

10300 Dual Drive Patches (MSD-SD2)

Rem: You "dual-ly" operators have to do a little work since I took some single drive short cuts. Pay close attention to "drive number." In this text you'll see a "0" or "0:" or none at all. If I specified the 0, you're fine. If I did not you'll have to do so in most cases. If you want to address drive number "one," the slot on the right, replace 0's with 1's.

The LOAD directory commands may cause some problems if you don't specify a drive number and you don't have a disk in one of the drives. Your "dual-ly" will try to read both directories for you.

10310 The following program lines must be changed or you'll wind up with some read error. Please accept my apologies for the inconvenience. (You lucky stiffs—you deserve the extra work).

{ "1ST SEQ FILE PGM" "SEQ FILE # PGM" "SEQ FILE PRINT" }	all require changes in	{ Line 300 Line 600 Line 800 }	Use "COMMAND DEMO," Lines 254 and 274 as your models.
----------------------------------------------------------------	------------------------------	--------------------------------------	----------------------------------------------------------

{ "COMMAND DEMO" }	may require a change in line 220.
--------------------	-----------------------------------

{ EXPS-PRINT }	replace "\$" with "\$0" in line 101.
----------------	--------------------------------------

{ GEN REL GEN REL PRINT }	each require changes in	{ Lines 320 and 20120 (see below) }
------------------------------	-------------------------------	----------------------------------------

```
320 OPEN3,8,3,"0:"+FF$+"L," +CHR$(L)
20120 OPEN3,8,3,"0:"+FF$
```

(Continued)

² - Use of «Wedge» may conflict with some machine language programs. Power off, wait 10 seconds, power on between loads.

³ - Computed from loader on TEST/DEMO DISK. "C-64 WEDGE." Agrees with that specified in "DISK BONUS PACK."

* - Use "0:" to decrease probability of DOS problems. The "replace option" was omitted on purpose. Change to "1:" for use of 2nd drive in a dual drive system. (10200)

10310 (Cont'd)

{ RANDOM FILES } Can be patched up by specifying the drive number in lines 100, 110, 120, 400, 410, -
{ RANDOM PRINT } 420 and 9000. By the time you get to this point you'll be able to handle the changes
yourself. Be sure to use 0 or the BLOCK commands will be foluled up.

10320 The «WEDGE» and DOS 5.1 works fine so long as you specify the drive number when needed. I didn't try everything but had no problems with what I did use. You'll need to get a copy. Hopefully your dealer can help.

10330 Be sure to check out the "DUPLICATE" command. Ignore all my remarks on back up programs and use this beauty when you want to copy an entire disk. Cover the write protect notch of your source disk.

10400 FORMS

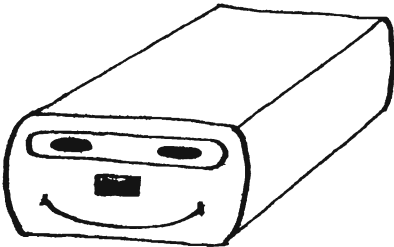
Rem: Feel free to copy for your own use. These forms may suggest some "programming" projects. Adapting "GEN REL" may be the way to start.

10410 HEAD CLEANING LOG



one per drive per year

HEAD CLEANING LOG		Drive _____	Year/19 _____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____



10420 **Hardware**

_____	_____		
Component	Serial Number		
_____	_____	_____	
Source	Phone Number	Date Acquired	
_____	_____	_____	_____
\$ Price	Tax	Other	Warranty Expires
_____	_____		
Total Cost	Paid by		

10430 **Repairs/Maintenance**

REPAIRS/MAINTENANCE _____

Component

DATE:	ACTION	SERVICED BY:	COST

May want to place on the back of 10420

10450 SOFTWARE

or

10451 Programs

TITLE _____	PGM _____
ON _____	_____
HARDWARE: Drive, Printer, Cassette, Joystick _____	
DATA FILES: _____	
PRELOAD _____	
LOAD _____	
PRE RUN _____	
RUN or SYS _____	

10452 General

_____	TYPE _____
ON _____	_____
CREATED BY _____	
PROGRAM _____	
FOR USE WITH _____	
PROGRAM/DATA FILE _____	
PROGRAM/DATA FILE _____	
PROGRAM/DATA FILE _____	

_____		SEQ/REL
STRUCTURE:	CONTENTS: _____	
NUMBER OF RECORDS _____	_____	
RECORD LENGTH _____	_____	
DATA TYPE: STRING/NUMERIC _____	_____	
NUMBER of FIELDS/RECORD _____	_____	
FILE NUMBERS _____, _____, _____		
NOTES:		

10500 ID CODE CHECK LIST

Simply circle used ID codes. Note duplicates² (if any) below.

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0G 0H 0I 0J 0K 0L 0M 0N 0O 0P 0Q 0R 0S 0T 0U 0V 0W 0X 0Y 0Z
 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 1G 1H 1I 1J 1K 1L 1M 1N 1O 1P 1Q 1R 1S 1T 1U 1V 1W 1X 1Y 1Z
 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 2G 2H 2I 2J 2K 2L 2M 2N 2O 2P 2Q 2R 2S 2T 2U 2V 2W 2X 2Y 2Z
 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 3G 3H 3I 3J 3K 3L 3M 3N 3O 3P 3Q 3R 3S 3T 3U 3V 3W 3X 3Y 3Z
 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 4G 4H 4I 4J 4K 4L 4M 4N 4O 4P 4Q 4R 4S 4T 4U 4V 4W 4X 4Y 4Z
 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 5G 5H 5I 5J 5K 5L 5M 5N 5O 5P 5Q 5R 5S 5T 5U 5V 5W 5X 5Y 5Z
 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 6G 6H 6I 6J 6K 6L 6M 6N 6O 6P 6Q 6R 6S 6T 6U 6V 6W 6X 6Y 6Z
 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 7G 7H 7I 7J 7K 7L 7M 7N 7O 7P 7Q 7R 7S 7T 7U 7V 7W 7X 7Y 7Z
 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 8G 8H 8I 8J 8K 8L 8M 8N 8O 8P 8Q 8R 8S 8T 8U 8V 8W 8X 8Y 8Z
 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F 9G 9H 9I 9J 9K 9L 9M 9N 9O 9P 9Q 9R 9S 9T 9U 9V 9W 9X 9Y 9Z
 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF AG AH AI AJ AK AL AM AN AO AP AQ AR AS AT AU AV AW AX AY AZ
 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF BG BH BI BJ BK BL BM BN BO BP BQ BR BS BT BU BV BW BX BY BZ
 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF CG CH CI CJ CK CL CM CN CO CP CQ CR CS CT CU CV CW CX CY CZ
 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF DG DH DI DJ DK DL DM DN DO DP DQ DR DS DT DU DV DW DX DY DZ
 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF EG EH EI EJ EK EL EM EN EO EP EQ ER ES ET EU EV EW EX EY EZ
 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF FG FH FI FJ FK FL FM FN FO FP FQ FR FS FT FU FV FW FX FY FZ
 G0 G1 G2 G3 G4 G5 G6 G7 G8 G9 GA GB GC GD GE GF GG GH GI GJ GK GL GM GN GO GP GQ GR GS GT GU GV GW GX GY GZ
 H0 H1 H2 H3 H4 H5 H6 H7 H8 H9 HA HB HC HD HE HF HG HH HI HJ HK HL HM HN HO HP HQ HR HS HT HU HV HW HX HY HZ
 I0 I1 I2 I3 I4 I5 I6 I7 I8 I9 IA IB IC ID IE IF IG IH II IJ IK IL IM IN IO IP IQ IR IS IT IU IV IW IX IY IZ
 J0 J1 J2 J3 J4 J5 J6 J7 J8 J9 JA JB JC JD JE JF JG JH JI JJ JK JL JM JN JO JP JQ JR JS JT JU JV JW JX JY JZ
 K0 K1 K2 K3 K4 K5 K6 K7 K8 K9 KA KB KC KD KE KF KG KH KI KJ KK KL KM KN KO KP KQ KR KS KT KU KV KW KX KY KZ
 L0 L1 L2 L3 L4 L5 L6 L7 L8 L9 LA LB LC LD LE LF LG LH LI LJ LK LL LM LN LO LP LQ LR LS LT LU LV LW LX LY LZ
 M0 M1 M2 M3 M4 M5 M6 M7 M8 M9 MA MB MC MD ME MF MG MH MI MJ MK ML MM MN MO MP MQ MR MS MT MU MV MW MX MY MZ
 N0 N1 N2 N3 N4 N5 N6 N7 N8 N9 NA NB NC ND NE NF NG NH NI NJ NK NL NM NN NO NP NQ NR NS NT NU NV NW NX NY NZ
 O0 O1 O2 O3 O4 O5 O6 O7 O8 O9 OA OB OC OD OE OF OG OH OI OJ OK OL OM ON OO OP OQ OR OS OT OU OV OW OX OY OZ
 P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 PA PB PC PD PE PF PG PH PI PJ PK PL PM PN PO PP PQ PR PS PT PU PV PW PX PY PZ
 Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 QA QB QC QD QE QF QG QH QI QJ QK QL QM QN QO QP QQ QR QS QT QU QV QW QX QY QZ
 R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 RA RB RC RD RE RF RG RH RI RJ RK RL RM RN RO RP RQ RR RS RT RU RV RW RX RY RZ
 S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 SA SB SC SD SE SF SG SH SI SJ SK SL SM SN SO SP SQ SR SS ST SU SV SW SX SY SZ
 T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 TA TB TC TD TE TF TG TH TI TJ TK TL TM TN TO TP TQ TR TS TT TU TV TW TX TY TZ
 U0 U1 U2 U3 U4 U5 U6 U7 U8 U9 UA UB UC UD UE UF UG UH UI UJ UK UL UM UN UO UP UQ UR US UT UU UV UW UX UY UZ
 V0 V1 V2 V3 V4 V5 V6 V7 V8 V9 VA VB VC VD VE VF VG VH VI VJ VK VL VM VN VO VP VQ VR VS VT VU VV VW VX VY VZ
 W0 W1 W2 W3 W4 W5 W6 W7 W8 W9 WA WB WC WD WE WF WG WH WI WJ WK WL WM WN WO WP WQ WR WS WT WU WV WW WX WY WZ
 X0 X1 X2 X3 X4 X5 X6 X7 X8 X9 XA XB XC XD XE XF XG XH XI XJ XK XL XM XN XO XP XQ XR XS XT XU XV XW XX XY XZ
 Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7 Y8 Y9 YA YB YC YD YE YF YG YH YI YJ YK YL YM YN YO YP YQ YR YS YT YU YV YW YX YY YZ
 Z0 Z1 Z2 Z3 Z4 Z5 Z6 Z7 Z8 Z9 ZA ZB ZC ZD ZE ZF ZG ZH ZI ZJ ZK ZL ZM ZN ZO ZP ZQ ZR ZS ZT ZU ZV ZW ZX ZY ZZ

Duplicates²

² - Use a back up program to change id codes without destroying disk contents.

10600 PRINT PROGRAM LISTINGS

10610 SEQ FILE PRINT

```

10 REM *** SEQ FILE PRINT ***
20 OPEN15,8,15:C%=CHR$(13):M%=",:OPEN4,4:C%=CHR$(13):M%=CHR$(44)
22 PRINT"(CLR)"
24 INPUT"FILE NAME";FF$
26 PRINT#4,C%$FF$C$
30 PRINT"(C/DN)"FF$C$C$"READ, WRITE, GET, CHANGE, OR QUIT"
40 GETK$:IFK$=""THEN40
45 IFK$="Q"THEN10001
50 IFK$="R"THEN600
55 IFK$="G"THEN800
60 IFK$="C"THEN400
65 IFK$("<"W"THENPRINT"R,W,G,C,Q":GOTO40
300 OPEN3,8,3,FF$+",S,W":GOSUB9999
310 PRINT"TO WRITE: "FF$C$
320 INPUT"NAME";N$
330 INPUT"TEST SCORE";T$
340 PRINT#3,N$M$T$
341 GOSUB9999
350 PRINT"ANOTHER?(Y/N)"
355 GETK$:IFK$=""THEN355
360 IFK$="N"THENPRINT#3,"*":CLOSE3:GOTO30
370 IFK$("<"Y"THEN350
380 GOTO320
400 PRINT"TO CHANGE: "FF$C$
405 PRINT"SHOULD (RVON)"FF$(RVDF) BE SCRATCHED??(Y/N)
410 GETK$:IFK$=""THEN410
415 IFK$="N"THEN24
420 IFK$("<"Y"THEN410
430 PRINT#15,"S0:"FF$:GOSUB9999:PRINTEM$
440 GOTO300
600 OPEN3,8,3,FF$+",S,R"
605 PRINT#4,C%FF$
610 PRINT"READING FILE: "FF$C$C$
620 INPUT#3,N$,T$
621 GOSUB9999:PRINT#4,N$,T$
625 ILEFT$(N$,1)="*"THENCLOSE3:PRINT#4,C%C$:GOTO30
630 PRINTN$TAB(25)T$
640 GOTO620
800 OPEN3,8,3,FF$+",S,R"
805 PRINT#4,C%FF$
810 PRINT"GET: "FF$C$
820 GET#3,G$
821 GOSUB9999
825 IFG%=CHR$(13)THENG$="←"
830 PRINTG$;:PRINT#4,G$;
835 IFG$("<"*"THEN820
840 PRINTC$:PRINT#4,C$
841 CLOSE3
842 GOTO30
9999 INPUT#15,EN,EM$,ET,ES
10000 IFEN<20THENRETURN
10001 PRINTEN;EM$,ET;ES:CLOSE3:CLOSE15:CLOSE4:END

```

10620 GEN REL PRINT

```

100 REM *** GEN REL PRINT ***
101 REM ***PRINTER REQUIRED***
110 READD
115 REM DIM DC$(D),L(D),P(D),D$(D)
120 FORI=1TOD
125 READDC$(I),L(I)
130 P(I)=L(I-1)+P(I-1)+1
135 L=P(I)+L(I)
140 D$(I)="-"
145 NEXT
150 PRINT{CLR}GEN REL PRINT":PRINT{C/DN}C/DN}PRINTER REQUIRED":GOSUB9500:OPEN4,4:R$=CHR$(13)
200 PRINT{CLR}FILE FORMAT
210 PRINT{C/DN}CATAGORY"TAB(20)"LENGTH POSITION{C/DN}"
220 FORI=1TOD
230 PRINTDC$(I)TAB(20)L(I)TAB(30)P(I)
240 NEXT
250 PRINT{C/DN}C/DN}RECORD LENGTH = "L"{C/DN}C/DN}"
255 IFL>254THENPRINT{C/DN}TOO LONG"
260 PRINT{C/DN} F1 - TO PRINT, C - CONT, ← - ESCAPE"
265 GETK$:IFK$=""THEN265
270 IFK$="←"THENEND
275 IFK$="C"THEN300
280 IFK$<>CHR$(133)THEN265
285 PRINT#4,R$R$"FILE FORMAT"R$R$"CATAGORY"CHR$(16)"20LENGTH POSITION"R$
290 FORI=1TOD:PRINT#4,DC$(I)CHR$(16)"20"L(I)CHR$(16)"30"P(I):NEXT
291 PRINT#4,R$"RECORD LENGTH = "L
292 PRINT{C/UP} C - CONT, ← - ESCAPE
293 GETK$:IFK$=""THEN293
294 IFK$="←"THENCLOSE4:END
295 IFK$<>"C"THEN293
300 OPEN15,B,15
310 INPUT"FILE NAME";FF$
315 PRINT#4,R$R$CHR$(14)FF$CHR$(15)R$
320 OPEN3,B,3,FF$+",L,"+CHR$(L)
321 GOSUB9999
325 PRINT{C/DN}ANY KEY TO CONTINUE"
327 GETK$:IFK$=""THEN327
330 PRINT{CLR}"
340 PRINTTAB(18){RVON}C/DN}C/DN}MENU{RVDF}":PRINT"FILE: "FF${C/DN}"
341 PRINT" 1 - CREATE NEW FILE
342 PRINT" 2 - EXPAND FILE
343 PRINT" 3 - WRITE IN SEQUENCE
344 PRINT" 4 - WRITE INDIVIDUAL RECORD
345 PRINT" 5 - READ/PRINT IN SEQUENCE
346 PRINT" 6 - READ/PRINT EXISTING RECORD
347 PRINT" 7 - EDIT EXISTING RECORD
348 PRINT" 8 - CHANGE FILE
349 PRINT" 9 - END RUN
350 IFM>0THENPRINT{C/DN}LAST JOB: "M

```

(Continued)

10620 (Cont'd)

```

355 PRINT"FILE: "FF$, REC #RN"(C/DN)"
360 INPUT"JOB #";M
370 IFM<10RM>9THEN360
380 IFM=8THENCLOSE3:GOTO310
390 IFM=9THENCLOSE3:CLOSE15:PRINT#4,R#R$:CLOSE4:END
400 DNMGOTO1100,1200,1300,1400,1500,1600,1700
910 DATA9
911 DATA LAST NAME,16
912 DATA FIRST NAME,12
913 DATA SOC SEC #,11
914 DATA CLASS(9-12),2
915 DATA SEX(M/F),1
916 DATA AVE(X.XX),4
917 DATA D.O.B.(XX-XX-XX),8
918 DATA PHONE#(XXX-XXXX),8
919 DATA COMMENT,9
1100 REM ** CREATE NEW FILE **
1110 PRINT"(CLR)(C/DN)(C/DN)(RVON)CREATE(RVDF): "FF$
1120 PRINT"(RVON)(C/DN)(C/DN)WILL DESTROY DATA IF USED ON OLD FILE(C/DN)(C/DN)"
1121 GOSUB9500
1130 PRINT"(C/DN)DIRECTORY ENTRY CREATED"
1135 PRINT"(C/DN)ANTICIPATED NUMBER OF RECORDS":INPUTM
1140 GOSUB9500
1160 P=1
1165 FORI=1TON
1170 RN=I
1175 GOSUB8888
1180 GOSUB7777
1185 PRINT#3,"I"
1186 GOSUB9999
1190 NEXT
1199 GOTO340
1200 REM ** EXPAND FILE **
1210 PRINT"(CLR)(C/DN)(C/DN)(RVON)EXPAND(RVDF): "FF$
1220 PRINT"(RVON)(C/DN)(C/DN)WILL DESTROY DATA IF USED ON OLD FILE(C/DN)(C/DN)"
1230 PRINT"(C/DN)CURRENT LAST RECORD #":INPUTS
1235 PRINT"(C/DN)NUMBER OF RECORDS TO ADD":INPUTA
1240 GOSUB9500
1260 P=1
1265 FORI=S+1TOS+A
1270 RN=I
1275 GOSUB8888
1280 GOSUB7777
1285 PRINT#3,"I"
1286 GOSUB9999
1290 NEXT
1299 GOTO340
1300 REM ** WRITE IN SEQUENCE **

```

(Continued)

10620 (Cont'd)

```

1305 PRINT{CLR}{C/DN}{RVON}WRITE{RVOF} IN SEQUENCE STARTING WITH"
1310 INPUT{C/DN}RECORD #";S
1315 INPUT{C/DN}WRITE TO #";A
1320 FORJ=STOA
1330 RN=J
1340 GOSUB2000
1350 GOSUB9500
1360 NEXTJ
1399 GOTO340
1400 REM ** WRITE INDIVIDUAL RECORD **
1410 INPUT"RECORD #";RN
1420 GOSUB2000
1430 PRINT{C/DN}{C/DN}ANY KEY FOR MENU"
1435 GETK$:IFK$=""THEN1435
1499 GOTO340
1500 REM ** READ IN SEQUENCE **
1505 PRINT{CLR}{C/DN}{RVON}READ{RVOF} IN SEQUENCE STARTING WITH"
1510 INPUT{C/DN}RECORD #";S
1515 INPUT{C/DN}READ TO #";A
1520 FORJ=STOA
1530 RN=J
1540 GOSUB4000
1550 GOSUB9500
1560 NEXTJ
1599 PRINT{CLR}":GOTO340
1600 REM ** READ INDIVIDUAL RECORD **
1610 INPUT"RECORD #";RN
1620 GOSUB4000
1630 PRINT{C/DN}{C/DN}ANY KEY FOR MENU"
1635 GETK$:IFK$=""THEN1635
1699 GOTO340
1700 REM ** EDIT RECORD **
1710 INPUT"RECORD #";RN
1715 GOSUB4000
1720 PRINT{C/DN}TYPE: 0 - NO CHANGE
1721 PRINT"    10 - CHANGE ALL
1722 PRINT"    1 TO 9 - FIELD CHANGE
1725 INPUTN:IFN<0ORN>10THEN1725
1730 IFN=0THEN340
1735 IFN=10THENGOSUB2030:GOTO1715
1740 PRINTDC$(N)
1745 INPUTD$(N)
1750 D$(N)=LEFT$(D$(N),L(N))
1755 PRINT"ANOTHER FIELD(0-9)":INPUTN
1760 IFN=0THEN1725
1765 IFN<10ORN>9THEN1755
1770 GOTO1740
1775 FORI=1TOD
1780 P=P(I)
1785 GOSUB7777
1790 PRINT#3,D$(I)
1791 GOSUB9999
1795 NEXT
1799 GOTO1715
1999 REM *** SUBROUTINES ***

```

(Continued)

10620 (Cont'd)

```

2000 REM ** WRITE RECORD **
2005 PRINT"(CLR)(C/DN)RECORD #"RN
2010 GOSUB8888
2011 GOSUB9100
2020 IFC$=""THEN2029
2025 PRINTRN" HAS BEEN WRITTEN"
2029 GOSUB9500
2030 FORI=1TOD
2035 P=P(I)
2040 PRINTDC$(I)" C "L(I)" SPACES ] "
2050 INPUTD$(I)
2055 D$(I)=LEFT$(D$(I),L(I))
2060 GOSUB7777
2070 PRINT#3,D$(I)
2071 GOSUB9999
2080 NEXT
2099 RETURN
4000 REM ** READ RECORD **
4010 GOSUB8888
4011 GOSUB9100
4015 IFC$=""THENPRINT#4,R#"#RN;C$:RETURN
4020 IFC$=""THENPRINT"(CLR)(C/DN)(C/DN)"RN" NOT WRITTEN":GOTO340
4025 PRINT"(CLR)(C/DN)"FF$TAB(20)"#RN"(C/DN)"
4030 FORI=1TOD
4035 P=P(I)
4040 GOSUB7777
4050 INPUT#3,D$(I)
4051 GOSUB9999
4060 PRINTI" "DC$(I)TAB(22)D$(I)
4070 NEXT
4080 GOSUB9600
4099 RETURN
7777 REM ** POSITION POINTER **
7780 PRINT#15,"P"CHR$(3)CHR$(LO)CHR$(HI)CHR$(P)
7781 GOSUB9999
7799 RETURN
8888 REM ** RECORD # ALGORITHM **
8890 HI=INT(RN/256)
8891 LO=RN-HI*256
8899 RETURN
9100 REM ** CHECK RECORD **
9110 P=1
9120 GOSUB7777
9130 GET#3,C$
9131 GOSUB9999
9140 IFC$=CHR$(255)THENPRINT"(CLR)(C/DN)(C/DN)"RN" NOT PRESENT":GOTO340
9199 RETURN
9500 REM ** WAIT ***
9510 PRINT"(C/DN)TYPE - C TO CONTINUE / ← ESCAPE TO MENU"
9520 GETK$:IFK$=""THEN9520
9530 IFK$="←"THENPRINT"(CLR)":GOTO340
9540 IFK$<>"C"THEN9520
9599 RETURN
9600 REM **** PRINT HARD COPY ****

```

(Continued)

10620 (Cont'd)

```

9610 PRINT"(C/DN) F1 - TO PRINT, C - CONT, ← - ESCAPE"
9615 GETK$:IFK$=""THEN9615
9620 IFK$="←"THEN340
9630 IFK$="C"THENRETURN
9640 IFK$(>CHR$(133))THEN9615
9650 PRINT#4,R#"#RN
9655 PRINT#4,D$(1),"D$(2)
9660 FORI=3TOD:PRINT#4," "D$(I):NEXT
9699 RETURN
9999 REM ## ERROR CHANNEL READ ##
10000 INPUT#15,EN,EM$,ET,ES
10010 IFEN<20THENRETURN
10020 IFEN=50THENIFM<30RM>7THENRETURN
10030 IFEN=50THENPRINTEM$,RN:GOTO340
10040 PRINTEN;EM$,ET;ES:CLOSE3:CLOSE15:CLOSE4
19999 END
20000 REM ### SECONDARY PROGRAM ###
20100 PRINT"(CLR)":OPEN15,8,15:OPEN4,4
20110 INPUT"FILE NAME";FF$:PRINT#4,FF$
20120 OPEN3,8,3,FF$:GOSUB21000
20130 INPUT"RECORD #";RN:GOSUB8888
20140 INPUT"LENGTH";L
20145 PRINT#4,"RN="RN" L="L
20150 PRINT#15,"P"CHR$(3)CHR$(LO)CHR$(HI)CHR$(1):GOSUB21000
20160 FORI=1TOL
20170 GET#3,6$:GOSUB21000
20175 IF6$=CHR$(13)THEN6$="←"
20180 IF6$=""THEN6$="※"
20181 IF6$=CHR$(0)THEN6$=" "
20185 PRINT6$;:PRINT#4,6$;
20190 IF1/40-INT(1/40)<.0001THENPRINT#4,R$
20199 NEXT
20200 PRINTCHR$(13)"(C/DN)(C/DN)ANOTHER RECORD?(Y/N)"
20210 GETK$:IFK$=""THEN20210
20220 IFK$="Y"THEN20130
20999 CLOSE3:CLOSE15:CLOSE4:END
21000 INPUT#15,EN,EM$,ET,ES
21010 IFEN<20THENRETURN
21015 IFEN=50THENPRINT6$:PRINT#4,6$CHR$(13):CLOSE3:CLOSE15:CLOSE4:END
21020 PRINTEN;EM$,ET;ES:CLOSE3:CLOSE15:CLOSE4:END

```

10630 RANDOM PRINT

```

10 PRINT{CLR}RANDOM FILES{C/DN}{C/DN}:OPEN4,4
15 INPUT"FILE NAME";FF$:PRINT"READ OR WRITE"
20 GETK$:IFK$=""THEN20
21 IFK$="W"THENGOSUB100
22 IFK$="R"THENGOSUB400
23 END
99 REM **** TO WRITE RANDOM FILE ****
100 OPEN15,8,15,"I":M$=CHR$(44)
110 OPEN10,8,10,"#"
120 OPENS,8,5,"@:"+FF$+",S,W":N=1:GOSUB9900
200 N$=STR$(N):INPUT"(C/DN){C/DN}DATA $";D$
210 PRINT#10,N$M$D$
220 GOSUB8000
230 PRINT"TRACK-"T,"SECTOR-"S
240 PRINT#5,N;M$;T;M$;S
250 PRINT#15,"B-W:"10;0;T;S
260 PRINT"ANOTHER(Y/N)
261 GETK$:IFK$=""THEN261
262 IFK$="Y"THENN=N+1:GOTO200
263 IFK$(">)"N"THEN261
270 CLOSE5:CLOSE10:CLOSE15
299 RETURN
399 REM **** TO READ RANDOM FILE ****
400 OPEN15,8,15,"I":C$=CHR$(13)
410 OPEN10,8,10,"#"
420 OPENS,8,5,FF$+",S,R":GOSUB9900
430 PRINT"WHICH RECORD TO READ?":INPUTR:PRINTC$ " #", " T", " S", "{C/LF}{C/LF}STATUS"C$
435 PRINT#4, " N", " T", " S", " ST"
440 INPUT#5,N,T,S:PRINTN,T,S,ST:X=ST
445 PRINT#4,N,T,S,X
450 IFST(>)THENPRINT"LAST RECORD IN FILE"
455 IFST(>)THENPRINT#4,"LAST RECORD IN FILE"
460 IFN(>)RNDST=0THENGOTO440
465 IFN=RTHEN500
470 PRINT"RECORD NOT FOUND":GOTO540
475 PRINT#4,"RECORD NOT FOUND"
500 PRINT"#N" TRACK-"T" SECTOR"S
505 PRINT#4,"#N" TRACK-"T" SECTOR"S
510 PRINT#15,"B-R:"10;0;T;S
520 INPUT#10,N$,D$
530 PRINTC$"#N$C$D$
535 PRINT#4,C$"#N$C$D$C$C$
540 CLOSE5:PRINT"(C/DN){C/DN}READ ANOTHER(Y/N)"
541 GETK$:IFK$=""THEN541
542 IFK$="N"THENCLOSE10:CLOSE15:RETURN
543 IFK$(">)"Y"THEN541
550 GOTO420
8000 REM*****BLOCK ALLOCATE*****
8010 T=1:S=1
8020 PRINT#15,"B-A:"0,T,S
8030 INPUT#15,EN,EM$,ET,ES
8040 IFEN=65THENT=ET:S=ES:GOTO8020
8050 RETURN

```

(Continued)

10630 (Cont'd)

```
8900 END
8998 REM*****GET SEQ FILE*****
8999 REM**      RUN 9000      **
9000 OPENS,8,5,"RND SAMPLE,S,R"
9010 GET#5,6#:PRINT6#;
9020 IFST=64THENCLOSE5:END
9030 GOTO9010
9900 REM***READ ERROR CHANNEL***
9910 INPUT#15,EN,EM#,ET,ES
9920 IFEN<20THENRETURN
9925 IFEN<20THENRETURN
9930 PRINTEN;EM#,ET,ES:CLOSE5:CLOSE10:CLOSE15:END
```


10700 SYMBOL LIST (Not including "normal" usage.)

SYMBOL	MEANING	EXAMPLE
*	multiplication	C=A*B
*	open file marker	. . . *SEQ
*	pattern match symbol	LOAD"*",8
/	division	X=Y/256
/	«Wedge» LOAD normal	/PROGRAM
\$	hexadecimal numbers	\$FF
\$	string variable	N\$
\$	string command	CHR\$() etc.
\$	name of directory	LOAD"\$",8
>	greater than	IFA>1THEN . .
>	«Wedge» substitute for @	>\$
?	abbreviation for PRINT	? "ABC"
?	wild card symbol	LOAD"A?C",8
%	integer variable type	%PROGRAM
%	«Wedge» LOAD special memory	N%
←	«Wedge» SAVE	←0: PROGRAM
#	abbreviation for "number" assembly language	#79
#	name of all random files	OPEN10,8,10,"#"
#	peripheral device signal for PRINT, INPUT, GET	PRINT#3,
@	«Wedge» command signal	@I
@	replace option signal	SAVE"0: PROGRAM,8"

10800 BIBLIOGRAPHY

Commodore 64 Programmer's Reference Guide; CBM, Inc. and Howard W. Sams & Co. Inc.; 1982, 1983.

Commodore 64 Software Bonus Pack; CBM, Inc.; 1983.

Commodore 64 User's Guide; CBM, Inc. and Howard W. Sams & Co. Inc.; 1982.

"Gazette Feedback," *COMPUTE!'s GAZETTE*; Issue 3, Volume 1, Number 3; September 1983.

"Gazette Feedback," *COMPUTE!'s GAZETTE*; Issue 3, Volume 1, Number 4; October 1983.

Microcomputer Interfacing; Pasahow, Edward J.; McGraw Hill Book Company/Gregg Division; 1981.

Microprocessors and Digital Systems; Hall, Douglas V.; McGraw Hill Book Company/Gregg Division; 1981.

VIC-1525 User's Manual; CBM, Inc.; 1982.

VIC-1541 User's Manual; CBM, Inc.; December 1982, September 1981.

10900 INDEX

A

activate wedge
1230,6215,6250
adaptations 7270
algorithm (see relative, record)
allocate blocks
4431
(see BLOCK-)
ASC or ASCII
5333,5411,6231,6347,6458,8500

B

[B-] (see BLOCK-)
backup
3320,3330,3350
BAM
0252,0271,2113,3110,3230,4431
4132,6349,6458,8120
bit 0236
blank disk
0250,2000,2103,2112,3122
block
0234,0235,0240,0261,0271,1113-1114
6347,6455,7122,8110,8120,8500
BLOCK-
ALLOCATE 8122,8340
FREE 8123
READ 8124,8330
WRITE 8125,8320
BUFFER-POINTER
8126,8320
byte
0236,0271,5344,6235,6347,6349,
7122

C

[C] (see copy)
capacity see storage
carriage return (see separator; chr\$(13))
[ch] (see channel number)
channel
5203,5204
command 4130,5312,7110,7300,
8320,8330
data 5312,5333,6141,6232,7141,7313
7334,8120,8320,8330
DOS 5312,6322,6347,6452
number 2313,4120
(see OPEN; secondary address;
error)

checklists

operation 0310,0410,0510,0610,0710
procedure 1010,2010,3010

CHR\$

(xx) 7140,7313
(0) 8500
(13) 5400,6231,6235,6263,7300
(32) 6347,850
(34) 6347
(44) 6231,6263,8320
(127) 8500
(255) 7336

CLOSE

5320
data file 5201,6140,6232,6234,6253
6342,7110,8127,8320,8330
(misc) 2213,2327,4121,4122,4130
4336,9234

colon(:)

6266

combine files see COPY

"COMBO ?" 6300

comma(,)

5442,6231,6234,6264

command channel

4120,4130,5312,6231,6233

"COMMAND DEMO" 6300

COMMANDS 4000

command

formats 1400,2400,3400,4120,4220
4320,4420,4520,4620,4720,4820
procedures 4120,4130
string 4120,5312,5333
(see GET#; INPUT#; OPEN; PRINT#;
random; relative; sequential)

connections

0314-0319

COPY, simple 4700

COPY to append

4800

6155,6325,6344

crunch

6230,6231,6340

D

data channel (see channel)

DATA FILES 5000

data files

command format (refer by type)
common procedures 5200
data format 5400,6260
(see also numeric; string;

10900 (Cont'd)

density 0284
 device number
 0150, 1222, 1400, 2312, 2344, 2400
 3233, 3400, 4120, 4134, 5212, 6231
 direct mode
 4120, 5344, 6148
 direction
 6120, 6130, 6232, 6234
 directory
 0082, 0260-0264, 1113, 1210, 1250,
 2113, 3110, 3220, 3350, 4210, 6322,
 6324, 6347-6349, 6400, 7131
 disk
 0200
 anatomy 0210-0230
 capacity see storage
 care of 0290
 insert 0500, 9610
 remove 0600, 9010, 9615
 (see blank ; format; ID; name)
 [dr] (see drive number)
 drive
 0100
 care of 0110
 dual 0150
 error see error
 ready 0143
 number 0150, 3360, 4120, 4834, 8120
 8320
 door 0512
 setup 0300
 maintainance 0110, 9640
 problems 9630, 9640
 operating system see DOS
 quiet 0140, 9410
 working 0144, 9420
 double
 density 0284
 sided 0285
 DOS 0101, 0103, 0262, 7120
 DOS 2.6 4120 (see COMMANDS; DOS)
 DOS 5.1 (see wedge)
 [dv] (see device number)

E
 edit (see relative; sequential)
 envelope, disk
 0210, 0292, 0500, 0600
 erase directory 4200
 erase disk (see FORMAT, NEW)
 erase file (see SCRATCH)

error
 channel 5203, 6141, 6236, 6250, 6342,
 6346, 7338, 9100, 9200
 (see command channel)
 condition 0145, 4131, 6140, 9410
 message 6236, 6253, 6324, 6342, 9103
 9212, 9232, 9250, 9500
 number 6236, 9232, 9500
 track/lector location 6236, 9232
 9253
 (see FLASHING RED LIGHT)
 essential procedures
 1200
 explanations, program
 6230, 6340, 6420, 7300, 8300, 8500, 9230
 "EX RANDOM" 8500
 "EXP\$-PRINT" 6400

F
 field
 7100
 file
 (see data, sequential, relative,
 random)
 "FILE ?" 6300
 file name
 0264, 6140, 7140
 file number
 2311, 2344, 4120, 5312, 5320, 5333
 5342, 6231-6236
 file type
 0261, 0272-0273, 6120, 6347, 6349
 6453, 7141, 7313
 (see types of)
 (first) "1ST SEQ FILE PGM"
 6200
 flag
 6232-6235, 6260, 7336
 FLASHING RED LIGHT 9000
 causes 6250, 9500
 emergency 9001-9005
 immediate mode 9300
 panic abort 9400
 problems 9600
 program method 9200
 remedies 9500
 misc 1117-1118, 1225, 1234, 1247, 1258
 2101, 3110, 3122, 3123, 3232, 6250, 8500
 (see error; wedge;)
 floppy development 0250
 (see disk)
 [fn] (see file number)

10900 (Cont'd)

"Friendly Floppy"0082,0210,1122,1349,2123,2205,
6215,8200**FORMAT 2000****format of**disk 0233,0271
general 1400,2400,3400,
text 0050
file data 5201,5400
(see **COMMAND**; **GET#**; **INPUT#**;
OPEN; **PRINT#**; **random**; **relative**;
sequential)**format (to)**how to 2210
process 0262,2110**format type** 0261-0262,0271**formatted disk** 0250,1122,3122**G****"GEN REL"**7200,7300
(see **adaptations**; **relative**;
job; **subroutines**; **variable list**)**GET# 5340**data files 6110,7110,8110
(misc) 5201,6142
samples 6235,6347,7336**"GR SAMPLE" 7240****H****hard disk** 0281**hard sector** 0283**header, directory**

0261,0271,6347,6349,6450

heads 0251,9640**headslot** 0220**hi (high byte) (see low byte)****"HOLD"** 6300**I****[I] (see INITIALIZE)****ID code**0081,0225,0252,0261-0262,2113
2113,2123,4231,4331,6341,6347**INITIALIZE 4300**

6234,6323,6341,8320,8330,9104,9303

index hole 0223**INPUT# 5340**data file 5400,6110,7110,8110
(misc) 5201,6142,9232
samples 6234,6236**J****jacket, disk**

0210,0291,0500,0600

jobmenu 6231,7314
descriptions 7220
selection 7315,7316**K****L****[L] 7141****label**

0225,0226,0292

length (see random; record; relative;
sequential)**light, green (see power)**

0120,0130,0293

light, red (see flashing)

0120,0130,0293,1100,9400

listing

backup 3340

program 6220,6330,6430,7400,8400,
8500**lo (low byte)**

7143,7144,7334,7335

LOAD 1000

type,normal 1262

M**menu (see job)****messages**

load 1110

verify 3310

save 3110,3242

scratch 4531

(see **command string**; **error**;
random; **relative**)**N****[N] (see NEW)****name**

disk 0261,0262,2122,4220

find 1360

file 0261,6140

program 0261,0264,1123,1220,3125,
3210,**NEW 2000,4200**

2216,2230,2320

10900 (Cont'd)

noises

0130, 0140, 0293, 1113, 2215, 9630

normal memory

1262, 3210, 3233

number (see channel; device; drive,
error; file; record)

numeric data 5450, 6261

O

objectives

program 6210, 6310, 6410, 7210, 8210
text 0040

OPEN 5310

data files 5201
experiments 2330
function of 5313
random file 8121, 8320, 8330, 8500
relative file 7110, 7141, 7142, 7313
sequential file 6100, 6232, 6234
use of 2212, 2310, 4130, 6251
(see COMMANDS;)

"OPEN EXP" 2331

P

[P] (see relative, position)

pattern matching 1340, 4532

position (see BUFFER; relative,)

power

light 0120, 0130
off 0293, 0700
on 0293, 0400

practice disk 6322

program

flow 7302
loss, alternatives 1370, 9234
mode 4120
(see explanations; file type;
listing; name; objectives; sample;
user instructions)

"PROGRAM NAME" 1220

program notes 0090

PRINT# 5330

data file 5201, 5400, 6140, 6260, 7110
function 5333
experiments with 2340
position (see relative)
punctuation 4132, 5440
use of 2216, 2320
(see printer; COMMANDS; BLOCK-)

printer

0020, 0300, 0700, 1374, 3340, 5312, 6422

punctuation (see format, data file;
PRINT#; separator)

Q

quick load 1310, 1320

quit wedge 1251

R

[R] (see direction; sequential, read;
RENAME)

RANDOM files 8000, 5120

advantages 5123
disadvantages 5124
error messages 8122, 8340
length 5122
nature 5121
overview 8110
read 8124, 8128, 8222, 8330
sequential 8211, 8220, 8320, 8330
validate 4431, 8123
write 8125, 8129, 8224, 8320, 8340
(see BLOCK-; BUFFER; track)

"RANDOM FILES" 8100

read (see error; GET#; INPUT#; OPEN;
sequential; relative; random)

record (see relative)

red light (see light; flashing; drive,
working)

RELATIVE files 7000, 5130

adaptations 7270
advantages 5133
create 7141, 7151, 7221, 7251, 7321
disadvantages 5134
edit 7227, 7257, 7327
error messages 7150
examine 7249, 7340
expand 7152, 7222, 7252, 7322
field 7100, 7133, 7261, 7273
file format 7242, 7270, 7311
length 5132
nature 5131
position 7110, 7143, 7260, 7334, 7341
read 7110, 7225, 7226, 7245-7248
7325, 7326, 7333
record 7100, 7120, 7130
record length 7131, 7261, 7273, 7313
record number 7132, 7264, 7273, 7335
record structure 7260, 7270, 7311
side sector 0273, 7120
write 7110, 7153, 7223, 7224, 7253
7254, 7323, 7324, 7332
(see "GEN REL"; program flow;
SCRATCH; secondary program;

RENAME 4600

6155, 6326, 6345

10900 (Cont'd)

replace option
 problems with 3350,6155,8320
 save with 3230,
 sequential files 6150

"RND SAMPLE"
 8200

RUN/STOP
 6253

S
 S (see file type; sequential; SCRATCH)
 sample programs
 6200,6300,6400,7200,7300,8200,8500

SAVE 3000

SCRATCH 4500
 3350,6150,6233,6326-6329,6346,7280

secondary address
 1262,1400,3220,3233,3400
 (see channel number)

secondary program
 7249,7301,7340,8223
 (see random; relative,examine)

sector
 0232-0233,0235,0240,0270,0283
 1113,1114,2113,7120,8110,8120
 8320-8340,8500,9250

semicolon(;) 5441,6265

separator, data
 5344,5400,6231,6234,6262-6266
 7260

"SEQ FILE # PGM" 6260

SEQUENTIAL files 6000,5110
 advantages 5113
 disadvantages 5114
 edit 6150
 length 5112
 overview 6110
 nature 5111
 read 6130,6140,6234,6252,6343
 using 6240,6250,6324
 write 6120,6140,6150,6232,6251
 6253,6342
 (see OPEN; GET#; INPUT#;
 PRINT#; format; random)

set up 0300

single
 density 0284
 drive 0150
 sided disk 0285

size, disk 0282

slick tricks 1300

soft sectored 0283

spanning 7264

special memory
 load 1262,1400
 save 3220,3233,3400

ST (see STATUS)

STATUS
 6344,6347,8330,8500

storage
 capacity 0240,2206,3123
 (see relative,record number;
 record length)
 of disks 0294

string data 6261
 (see variable; command; message)

subroutines
 7302,7330,8340-8360

system 0020
 (see setup)

T
 tape backup 3330
 "TEST 1" 6240,6250

track
 0231,0233,0240,1113-1114,2113
 8110,8120,8320-8340,8500,9250

track 18
 0261,0270,6349,6458,9253

types of files 5100
 (see file type)

U
 U1 (or UA) (see USER1)
 U2 (or UB) (see USER2)
 UI 9004,9302

unclosed file mark
 0261,4434

user instructions
 program 6215,6240,6320,7230,7240
 8220,9240
 text 0010-0090

USER 1 (or USER A)
 8128,8500

USER 2 (or USER B)
 8129

V
 [V] (see VALIDATE)
 VALIDATE 4400,8123
 (see random)

10900 (Cont'd)

variable

file name 6232
"GEN REL" list 7301
lists 5343
numeric 5344,5450,6261
program names 1350
string 5344,5400,6232,6261

VERIFY 3300,3404

view directory

1250,3122,6250,6322,6347

W

[W] (see direction; sequential,write)

wait routine 7337

wedge

activate 1230,9621,9622
check 9102
deactivate 1251
problems 9620
read error channel 9100,9254
use of 1240,2233,3240,4120,4433
6215,6250,6322

(see COMMANDS; error)

wild cards 1330,4532,6322

write protect notch 0081,0221,3110

write (see PRINT#; random; relative;
sequential)

X

Y

Z

zero (0) 0000

(see drive number)

FRIENDLY FLOPPY NW

DISK DRIVE GUIDE

© Copyright 1984, Con Cor and Nancy Wilmot



C-64™

DISK DRIVE GUIDE

By NANCY L. WILMOT

- ★ Sample Programs
- ★ Sample Files
- ★ Tutorials
- ★ Check Lists
- ★ ...and a lot more!

Disk Included

DISK DRIVE GUIDE assumes no prior knowledge of disks, drives or disk drive "jargon," (DOS, BAM, etc.).

If you have no desire to program, you'll be more comfortable running your software if you can communicate directly with your drive. You'll gain knowledge and experience playing with the sample programs. There is even a sample RELATIVE file program you can adapt to your own needs without being able to program.

If you have a little knowledge of BASIC and want to write, (or simply better understand), file management programs, you'll find detailed treatments of SEQUENTIAL, RELATIVE and RANDOM files. File operations in the sample programs are explained, line-by-line.

"Friendly Floppy," the disk included, contains the sample programs and files. You won't have to waste hours and hours typing and de-bugging.

If you want to learn to write machine language routines, program the controller, or repair your equipment, look for another source. (Sorry, but I don't want to waste your time or money). If you are looking for a stiff, formal, technical approach you won't find it here. I only use that style when absolutely necessary and even then I can't maintain it for long. I do not subscribe to the philosophy that solid learning can't be fun.

If you must have a promise of **all about, easy, complete** or any of the other expletives, I can't give it to you. All I can promise is that there is a lot of material here for all but the advanced data file programmer. Many topics are easy, some are not, and all become easier once you know how.

Disk included... Sample programs... Sample files... Illustrations... Charts... Tutorials... References... Check lists... Definitions... Examples... Flashing Red Light... User instructions... Programs in BASIC and completely accessible for user alterations... Program listings... Error conditions as tools... How to... Why to... When to... What not to... Extensive cross references... Detailed Index... Care and feeding of disks and the drive... Handy forms for keeping track of your hardware, software and costs...

SUPPORTS:

C-64™, 1541™, SX-64™ and MSD-SD1™, MSD-SD2™ ...

Printer Optional ...

VIC-20™ and 1540™/1541™ user's please note: The methods of this text apply. Some programs may exceed memory capacity and some screen displays may need alterations.

PUBLISHED BY CON-COR INTERNATIONAL, 1025 INDUSTRIAL DRIVE, BENSENVILLE, IL 60106-1297

