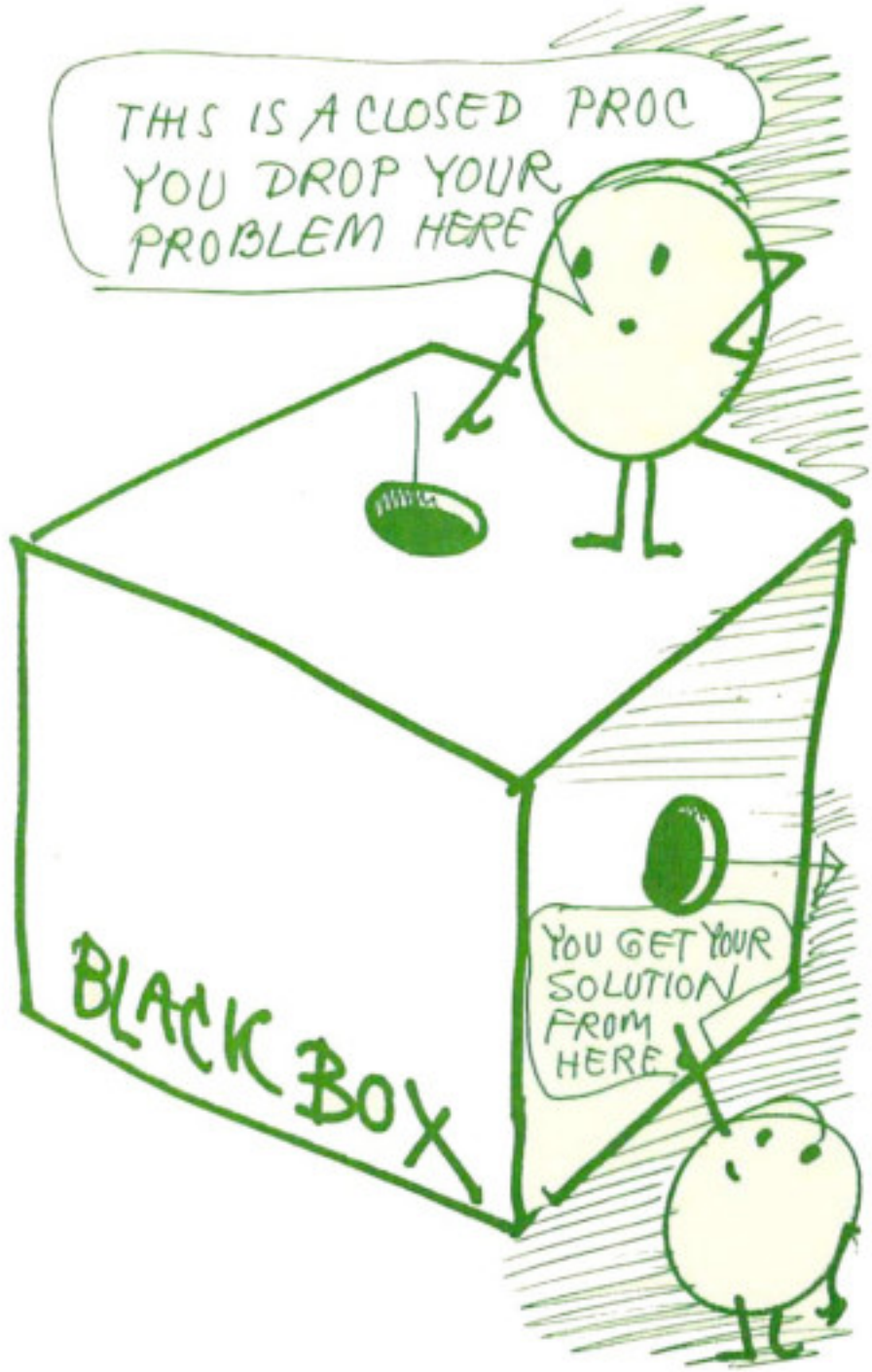
THE AMAZING ADVENTURES OF CAPTAIN COMAL BOOK 1

CAPTAIN COMAL GETS ORGANIZED



HOW MODULAR PROGRAMMING MAKES PROGRAM CONSTRUCTION EASIER
SHARE DATA FILES :: SHARE MODULES
DISK ORGANIZATION SYSTEM INCLUDED ON DISK - READY TO RUN

CAPTAIN COMAL GETS ORGANIZED
UPDATE NUMBER 1

PROGRAM ADDITIONS

If you write any programs that work together with this Disk Organization System, we would be happy to share them with the many other users. Each additional program will be included on a COMAL TODAY disk and explained in the COMAL TODAY newsletter. If enough programs are submitted, a separate disk with these programs will be made available. We are pleased to report that David Stidolph is already completing a program to add to the system. His program can protect a file from being DELETEd, restore a deleted file, delete files, change a file name, change a file type, and alphabetize the directory. We wish David luck in completing this useful program and look forward to adding it to our collection. It will be included on the next available COMAL TODAY disk after we receive the program and test it.

UPDATES TO PROGRAMS

Any updates to the programs in this book / disk will be printed in the COMAL TODAY newsletter. This first update will not be in COMAL TODAY, since it is included with the book. The programs on the disk already include ALL of the updates shown. However, the book was already printed, and these updates are not included in the book. Thus, this is merely for your information. No need to update any programs on the disk.

PROGRAM :: STARTUP:

We added checking if a Master disk was formatted, and if so omit asking for the Master to be inserted:

```
From: FORMAT'MASTER
To:   FORMAT'MASTER(FORMATTED)
Add:  IF NOT FORMATTED THEN
Add:  ENDIF   after the PRINT "OK"

From: PROC FORMAT'MASTER CLOSED
To:   PROC FORMAT'MASTER(REF FORMATTED) CLOSED
Add:  FORMATTED:=FALSE

From: FORMAT'IT
To:   FORMAT'IT(FORMATTED)

From: PROC FORMAT'IT CLOSED
To:   PROC FORMAT'IT(REF FORMATTED) CLOSED

Just before WHEN "Q","q"
Add:  FORMATTED:=TRUE
Then From: NULL
     To:   FORMATTED:=FALSE
```

PROGRAM :: DOS'MENU:

In procedure DUAL'CHOICE add write protect check:

```
Add to DIM: , S$ OF 2
After the DELETE statement add:
S$:=STATUS$
IF S$="26" THEN
 FOR X=1 TO 30 DO PRINT CHR$(20),
 INPUT "TAKE OFF WRITE PROTECT TAB": S$
 CLOSE FILE 78
 PASS "I0" // CLEAR STATUS
ELSE
 WRITE FILE 78: DUAL'STATUS //already in program
ENDIF
```

PROGRAM .. MASTER'MAKER:

Add file count for master file maximum:

```
Add at beginning of program:
MAX'FILES:=3000; FILE'COUNT:=0

In procedure PROCESS'MASTER:
From: ADD'DIRECTORY(ADD'ID$,SHOW'DIR)
To:   IF FILE'COUNT<MAX'FILES-144 THEN ADD'DIREC
      TORY(ADD'ID$,SHOW'DIR)  all on one line
Just before the ENDPROC add:
PRINT FILE'COUNT;"FILES IN MASTER FILE."

In procedure ADD'DIRECTORY:
Just after the WRITE FILE ALL'FILE statement:
Add:  FILE'COUNT:+1
```

PROGRAM :: COMPARE'DIR:

In the DIMs change MAX'FILES to MAX'FILES+1

```
In procedure GET'DIR2:
After READ FILE DIR'FILE: COUNT
Add:  IF COUNT>MAX'FILES THEN COUNT:=MAX'FILES
```

>>>>> IMPORTANT INFORMATION <<<<<

HOW TO USE THE DISK ORGANIZATION SYSTEM:

Do NOT use the enclosed disk. Make a copy of it, store the enclosed disk in a safe place, and use the COPY. Instructions on how to make a disk copy are enclosed. Label the copy of the enclosed disk DOS MENU PROGRAM DISK.

You will need one more blank disk in addition to the backup copy you just made. The SYSTEM will format it for you. It will be labelled MASTER DIRECTORY DISK.

>>> Do NOT put write protect tabs on disks.

STARTUP:

Turn off your computer, then back on.
Insert the DOS MENU PROGRAM DISK into the disk drive and enter the following using unshifted letters (followed by the RETURN key):

```
LOAD "BOOT*",8
```

When the word READY. appears use unshifted letters to enter the following:

```
RUN
```

That's it. To restart the system at any time, from within COMAL enter the following command:

```
CHAIN "DOS'MENU"
```

MENU:

The SYSTEM will present a MENU for you. This makes it easy to use. The first time you use the SYSTEM you MUST choose option A to Allocate new files. You may also choose H for HELP at any time. The SYSTEM will format a new blank disk for you to use as the MASTER DIRECTORY DISK.

Follow the prompts provided. Once you have the files allocated, the next thing you should do is Update the system by cataloging your disks. Choose option U to do this. It will ask you to insert a disk to be cataloged. Try the DOS MENU PROGRAM DISK. It will read its directory, then ask you to insert the MASTER DIRECTORY DISK. Take out the disk and put in the MASTER disk. The system will catalog the directory of the previously read disk. Then it will ask you to insert another disk to catalog. Catalog a few more of your disks (you can finish cataloging all your disks later). When done, reply Q for Quit. You then are sent back to the MENU.

Now, choose option M to make a MASTER file from all the directories you just cataloged. The SYSTEM reminds you to choose option M anytime it is needed. Once the system makes the MASTER file, you will automatically be sent back to the MENU.

REVIEW:

You first loaded COMAL.
Then you Allocated files.
Then you Updated directories.
Then you Made the Master File.

NOW:

You now can explore the other options available:

C - Compare two directories. You can now compare any two directories that are cataloged. You must know the ID's of each directory (option S can help you recall the ID's).

D - Delete any directory from the Master (in case you lose the disk - or reformat it).

F - Find all the disks that contain a specific program. Or use the * for wildcard searching.

I - ID Chart or List. This is an easy way to determine all the ID's currently used so you can use unique ID's on the next disk's you format.

N - Number of drives. If you switch from a single drive to a dual drive, this option allows you to inform the SYSTEM of the upgrade. (Two single drives are not supported).

P - Print any directory from the Master. The directory can be printed as an expanded single column. Or print it in multiple columns. Printing can be either to the screen or the printer.

S - Summary of all disks in the Master. This is a quick overview of your disks, including the date cataloged, number of files, free blocks, disk name and disk ID.

V - View a directory of any disk. This option allows you to see a directory of any disk without having to catalog it onto the Master. The directory can be printed to your printer if you wish.

Q - Quit. Quit the system with this option.

That's it. The SYSTEM is self prompting, so have fun. The accompanying book shows how the programs were constructed. You can write your own programs to add to the SYSTEM.

TABLE OF CONTENTS

ISBN 0-928411-01-X

BOOK 1
CAPTAIN COMAL GETS ORGANIZED

Welcome to the first of our many amazing adventures. This series of books will show you how programming in COMAL can be exciting and fun. Just follow along with me and see how easy it can be. But no skipping pages. Read this book next to your computer with COMAL ready.

The first step is to transform your primitive BASIC Commodore computer into a powerful COMAL computer. That is easy! Everything you need is on the disk that came with this book. Do NOT use the original disk. See APPENDIX A for instructions on how to make a backup copy of it. Label the copy DOS MENU PROGRAM DISK. Once you have a copy of the disk you are ready to go on.

1) Turn on your computer and disk drive. If the computer was already on, turn it off and then back on.

2) Put the disk into the disk drive.

3) Type in the following line - remember to hit the RETURN at the end to inform the computer that you are done with the line:

>>> Always use unshifted letters with COMAL commands and statements <<<

```
LOAD "BOOT*",8
```

Your disk should start turning and soon your computer will be ready for its transformation into a powerhouse.

4) When the computer is done loading the boot file, it will say "READY.". Now type in the following line - remember the RETURN key at the end.

```
RUN
```

The CAPTAIN COMAL welcome should cover your screen. Then your disk will start turning again. After a couple minutes your computer's transformation from BASIC into COMAL will be complete. You are then ready to begin our first amazing adventure.

Most of the programs and exercises from this book are on the disk. You can be lazy and use them. But you will learn alot more by getting out a

new disk (properly formatted) and typing in everything yourself. It's not much typing, and you will enjoy some of the tricks and shortcuts I'll be showing you along the way. See APPENDIX B for instructions on how to format a disk.

GETTING STARTED

Our mission is to design, program and test a series of programs for organizing disks. It may sound rough - ten long programs! But by taking advantage of modular programming (made easy with COMAL) we can do it in a snap.

First, I better explain what a "module" is. A module is a number of program lines that perform a specific task.

A module may also be referred to as a routine, subroutine, procedure, or function. The best way to understand this concept is to create a simple module.

All our programs should start out with a greeting of some kind, so let's design a welcome module. This same module can be used with all of our programs in this book.

First, before you start typing on the computer, pull out some paper and a pencil (pencils are better than pens, since you can erase mistakes and change things around as your ideas get better and better). Now, lets design our first module.

Write down everything we should do to start off all our programs:

1. Say "Hello, welcome to organization."

Good start. Hmmmm. Maybe we should clear the screen first:

1. Clear the screen.
2. Say "Hello, welcome to organization."

Yes, that's better. But what about screen colors? Don't assume the current colors are the ones you want. It is easy with COMAL to set up any colors you want:

1. Clear the screen.
2. Set the screen colors.
3. Say "Hello, welcome to organization."

Now, you see why we need a pencil - and a big eraser! We changed our mind twice already, and hardly have started. Mind you - we did NOT make mistakes! Just improved an idea.

Now, where were we. Yes, we just provided a warm welcome.

Well, that is enough! It is always nice to keep our modules small. They are easier to maintain that way, and easier to combine with others as well (some people even say that if the module can't fit on your screen, it is too big - but they had 80 column screens!). So, our design is done. Now, lets write it in COMAL.

"Slow down!!!" I hear you say. OK! Remember, COMAL likes to do alot of the work for you. It will provide line numbers for you. It also will provide nice indenting of your program when you list it, automatically. But COMAL does not like it when you squish words or commands together. That may be OK for BASIC. But be civilized! Programs are to be read. They are not code. "OK OK!" you say. "Enough lectures!".

First tell COMAL to provide you with the line numbers:

```
AUTO
```

Remember to hit the RETURN key after every line. And, don't use the SHIFT key when typing in commands and COMAL statements. They are listed in UPPER case only to help you distinguish the COMAL lines from my story lines. After you typed AUTO and hit RETURN COMAL responds with your first line number:

```
0010
```

COMAL will continue providing you line numbers (0020, 0030, 0040, ...) as you type in the following program instructions:

```
PRINT CHR$(147),                        clear the screen
BACKGROUND 0                            0 means black
BORDER 12                               12 means medium gray
PENCOLOR 12
PRINT "Hello, welcome to organization."
```

That's it. Yes, I know. COMAL gave you another line number, and we don't need it. Just hit return again (like a blank line) and COMAL will know that we are done with its automatic line numbers. Now, lets have COMAL show us what it thinks our program looks like (hopefully we typed it in right!):

LIST

After you hit the RETURN key COMAL printed the following on the screen:

```
0010 PRINT CHR$(147),
0020 BACKGROUND 0
0030 BORDER 12
0040 PENCOLOR 12
0050 PRINT "Hello, welcome to organization."
```

What! You can't get lower case! Ooops! We forgot that the Commodore computer has two modes, graphics and lower case. We must make sure the computer is in lower case mode everytime we run our program.

So, we found out by trying to implement our welcome module that our original design could be improved. But don't worry. All is not lost. Simply pencil in the following line on your design paper right between 1. and 2.:

1a. Set the computer into lower case mode.

Don't worry. COMAL is smart. You don't have to start over just because we changed our design. Let's just add a line in our program between line 10 and line 20 (now you know why COMAL counted by 10's and skipped some numbers). Type in the following line:

```
15 PRINT CHR$(14), // lower case mode
```

COMAL will let us put comments into our COMAL program as long as we start the comment with //. So, to remind ourselves later what we are doing in this line, we added the comment: // lower case mode.

Now, list the program again:

```
0010 PRINT CHR$(147),
0015 PRINT CHR$(14), // lower case mode
0020 BACKGROUND 0
0030 BORDER 12
0040 PENCOLOR 12
0050 PRINT "Hello, welcome to organization."
```

That line 15 sticks out like a sore thumb. Later everyone will think we added it after we wrote line 10 and 20 (which we did, but why should they know?). So lets tell COMAL to renumber the lines for us:

```
RENUM
```

Hit RETURN and in a split second, COMAL has renumbered the lines. Check and see for yourself:

```
LIST

0010 PRINT CHR$(147),
0020 PRINT CHR$(14), // lower case mode
0030 BACKGROUND 0
0040 BORDER 12
0050 PENCOLOR 12
0060 PRINT "Hello, welcome to organization."
```

Now that's better. I know you want to quick RUN the program to see if it works nicely. But, have patience. First SAVE it on disk. It is a good guideline to always SAVE a program before you RUN it. Who knows, the power may fail, or you may forget to SAVE it later. So, let's SAVE the program now:

```
SAVE "WELCOME1"
```

We called it "WELCOME1" instead of "WELCOME", since we may be changing it further (next will be "WELCOME2", then "WELCOME3", ...). When we are all finished and it is tested and working, then we will save it as "WELCOME".

Now, we can run our program:

```
RUN

Hello, welcome to organization.
```

Wow! It worked. Let's see it again. OK:

```
RUN

Hello, welcome to organization.
```

Nice! Everytime we say RUN it will do exactly what we told it. Do we like the colors? It's easy to change. Let's try a pencolor of 15 instead of 12. To do this we simply edit the program. COMAL includes a FULL SCREEN editor that makes it very easy to edit the program line. If the line is on the screen - you can edit it. To get a line on the screen just LIST it (as we have already done before). Or, you can use the command EDIT instead of LIST. This will be helpful later when we have

longer programs:

```
EDIT

0010 PRINT CHR$(147),
0020 PRINT CHR$(14), // lower case mode
0030 BACKGROUND 0
0040 BORDER 12
0050 PENCOLOR 12
0060 PRINT "Hello, welcome to organization."
```

First move your cursor up to line 50. Now move the cursor over to the 2. Simply type a 5. The twelve is now a fifteen. The line looks changed. But COMAL doesn't actually change the line until you hit the RETURN key. It lets you reconsider or make further changes before you decide to finalize the change. So, hit the RETURN key now. Then:

```
LIST

0010 PRINT CHR$(147),
0020 PRINT CHR$(14), // lower case mode
0030 BACKGROUND 0
0040 BORDER 12
0050 PENCOLOR 15
0060 PRINT "Hello, welcome to organization."
```

It worked.

Now try out the program again. Do you like the change? If not, change it again. I like it. But we're not done yet. What we have now is a program. What we really want is a module. COMAL lets us name modules and treat them as a procedure or a function. The module we just programmed is a procedure. In order to be a function, it must return a value (more about that later). So lets name our procedure. WELCOME would be an appropriate name. (The name can be anything up to 16 characters, including letters, digits, apostrophe ('), square brackets ([]), backslash (\), and backarrow (<-) - the backarrow is converted into an underline by COMAL when listing the name to a printer) -- only the first character must be a letter.

So here are the lines to add:

```
5 PROC WELCOME
70 ENDPROC
```

That marks the start and end of our procedure. List it and see:

```
LIST

0005 PROC WELCOME
0010  PRINT CHR$(147),
0020  PRINT CHR$(14), // lower case mode
0030  BACKGROUND 0
0040  BORDER 12
0050  PENCOLOR 15
0060  PRINT "Hello, welcome to organization."
0070 ENDPROC
```

Now, run our procedure:

```
RUN
```

What? It doesn't work anymore? Good thing you saved it first (You DID remember to save it first, didn't you! I shouldn't have to remind you every time you know!). You probably saved it with:

```
SAVE "WELCOME2"
```

But, no cause for alarm. The procedure is working just fine. However, a procedure is only executed when it is called. So far, all we did is define it. To actually see it work simply call it by name (you can do this in direct mode):

```
WELCOME

Hello, welcome to organization.
```

The procedure was executed. Nice work! Now, anytime you type in WELCOME, COMAL will remember to execute your procedure named WELCOME.

I changed my mind. I think I like the PENCOLOR better as 12. Here is how to edit it (I remember that it is line 50):

```
EDIT 50

0050 PENCOLOR 15
```

Now simply move the cursor up on top of the 5 of 15 and type in a 2 to make it 12 again. Hit RETURN and that's it.

I think we have our final procedure: designed, written, and tested.

"WAIT", I hear someone say. They then start lecturing me on how to properly start a program. They said we can't just start every program saying exactly the same thing. How will the user know which program is running?

I agree. But we don't have to start over. We just modify what we have.

But how can we INDIVIDUALIZE the module so it works, UNCHANGED, with ALL our programs. A dilema. But don't lose hope. COMAL is a very capable language. The solution to this problem is one of COMAL's specialties - parameters.

We can pass a parameter to any procedure (and the procedure can pass one back to the program as well). One way to individualize the introduction to each program would be to print a TITLE on the top of the screen. The module would always print a title, but the title could be different each time, indicating what program was running - depending on the parameter passed into the module.

The parameter in a procedure can be an array or a simple variable. It can be a string, integer or real number. There can be one parameter, or many parameters - limited only by the 80 column line length. For now we will only be using one string parameter. So, let's modify our WELCOME procedure to become the more individualized INTRO procedure. First, list the procedure as it now stands:

```
EDIT

0005 PROC WELCOME
0010 PRINT CHR$(147),
0020 PRINT CHR$(14), //lower case mode
0030 BACKGROUND 0
0040 BORDER 12
0050 PENCOLOR 12
0060 PRINT "Hello, welcome to organization."
0070 ENDPROC WELCOME
```

Notice that the last line ENDPROC statement now has a name after. We neglected to put the name of the procedure that was ending with the ENDPROC, so COMAL did it for us.

Most of the procedure can stay the same. We only need to change the top line (called the HEADER), the message printing line (0060) and match the new procedure name in the ENDPROC (0070).

First, let's create a new HEADER. If we enter a new line numbered 5 it

will replace the line 5 that exists now (just what we want).

Enter this new line:

```
5 PROC INTRO(TITLE$)
```

TITLE$ is the string paramater. It is assigned the value of whatever string is used in the statement that calls this procedure. We can use it in line 60:

```
60 PRINT TITLE$
```

And fix the last line too:

```
70 ENDPROC INTRO
```

Now, list our modified module:

```
LIST

0005 PROC INTRO(TITLE$)
0010  PRINT CHR$(147),
0020  PRINT CHR$(14), //lower case mode
0030  BACKGROUND 0
0040  BORDER 12
0050  PENCOLOR 12
0060  PRINT TITLE$
0070 ENDPROC INTRO
```

OK, looks good. First SAVE it on disk (of course):

```
SAVE "INTRO1"
```

Now run it:

```
RUN
```

Nothing happens. Of course, we have to call the procedure. Try this:

```
INTRO("TESTING")

TESTING
```

It worked. Any string value passed to INTRO is printed at the top of the screen. Try another:

```
INTRO("CAPTAIN COMAL WAS HERE")

CAPTAIN COMAL WAS HERE
```

We now can store its final version on disk. But first lets renumber it to start with line 9000. This will make it easy later to merge this procedure with another program already in the computer (the line numbers are important during this merge). Since our programs will be numbered beginning with 10, the lines numbered over 9000 will not conflict with the current program. More on this later.

```
RENUM 9000
```

We also must know how to store this procedure on disk so we CAN merge it later. If you SAVE it to disk, all you can do with it later is recover it with a LOAD command. This does not allow you to merge procedures. COMAL has a second way to store a procedure on disk: LIST it to disk. Easy! Just add a file name after your list command, and the procedure will be listed to disk instead of your screen. Try it:

```
LIST "INTRO.L"
```

NOTICE! All our procedures and functions we store on disk with the LIST command will have their file name end with .L to remind us. To merge a procedure that was listed to disk use the ENTER command:

```
ENTER "INTRO.L"
```

But lets have a final review of our first procedure, now that we have RUN it, executed it, and renumbered it in the 9000's:

```
LIST

9000 PROC INTRO(TITLE$)
9010   PRINT CHR$(147),
9020   PRINT CHR$(14), // lower case mode
9030   BACKGROUND 0
9040   BORDER 12
9050   PENCOLOR 12
9060   PRINT TITLE$
9070 ENDPROC INTRO
```

If you have a printer, I'll show you how easy it is to list this procedure on your printer. Simply SELECT the printer as the OUTPUT location:

SELECT OUTPUT "LP:"

The LP: means Line Printer. The colon must be included. If you want you can omit the word OUTPUT, since COMAL will add it for you. Once you enter this line, anything normally printed on the screen will be printed on the printer (with some exceptions, such as error messages, input prompts, and disk catalogs via CAT). To see for yourself, now enter this:

PRINT "TESTING"

The word TESTING was printed on the printer. Now enter:

LIST

The procedure now printed on your printer. COMAL automatically returns the output to the screen after a LIST command. You can do this yourself if you wish like this:

SELECT OUTPUT "DS:"

This selects the Data Screen as the output location.

DESIGNING OUR DISK MANAGEMENT SYSTEM

Now, on with designing our Disk Management System (DMS). First, let's review just what we want the DMS to do. Then we can design a structure for any disk files it will use.

1) The DMS should be able to catalog a whole set of diskettes onto a MASTER DISK. Thus it will have to be able to read a disk's directory and store it on disk for future use.

2) The DMS should be able to list all the disk ID's currently used. Then you will be able to give a new disk an ID that will not conflict with one already in use.

3) The DMS should be able to tell you what disk's contain a specific program you are looking for.

4) The DMS should be able to quickly list all the programs on any specific disk. This list should also be easily useable in a program that would compare the directory of two different disks.

5) The DMS should be able to delete a disk from the MASTER. If any programs are added to a disk already cataloged on the MASTER, the DMS should be able to UPDATE the MASTER to reflect the current status of that disk.

6) The DMS should be able to list a summary of all the disks in the MASTER. It should know the date that each disk was cataloged on, how many files are on the disk, and the number of free blocks on the disk, as well as the disk name and ID.

All the data will be stored in data files on our MASTER DISK. It is important that we get our file structures right the first time. Try to look ahead and envision what we possibly will use the files for. It will create alot of extra work later if we find that we must change the structure of a file. All the programs that use it must then be modified to reflect this change. And there then could be side effects to the modifications. So let's think carefully about our files.

DESIGN A FILE STRUCTURE

We want to be able to read a disk's directory and write a copy of it on our MASTER DISK as fast as possible (1). Simply duplicating each disk's directory as a separate file would work. The file name could be the word DIRECTORY followed by the ID. Or adding two dots .. in between DIRECTORY and the ID would be more readable: DIRECTORY..ID.

But we also want to be able to scan through all the file names quickly to find which disk contains a specific program (3). Maybe we should write the directory of each disk one after another as one big file.

But we want to be able list all the disk ID's currently in use (2). That would be hard to do quickly with a big file of ALL the directories. So we may be best off creating an extra file of JUST THE DISK ID's.

But we also must be able to produce a summary of all disk's in the system (6). We could create a special file for this as well. But then it would be difficult to delete or update a disk. If each disk had its own file for its directory, and stored the disk's summary information at the start of the file, we could just open each disk directory file, read the summary info, print it, close the file and go on to the next directory file. And if we had a file of just disk ID's we would know exactly what directory files were on the disk to look at.

Since we want to be able to list all the files on any specific disk at

any time (4), having the complete directory as a separate file will make that easy. It would also be easy to delete or update a disk's directory (5) since it was a separate file.

What files to have seems to be just as important as how to organize the files. Indeed, I spent several MONTHS contemplating the question of what files to have and how to structure them. But we don't have time to go into all the gruesome details. I will just provide you with the final file design along with explanation. It will work well with all the programs in the DMS and is easily adaptable to any DMS extensions you may write.

MASTER DIRECTORY DISK FILES STRUCTURE

DIRECTORY FILE

FILE USE : Directory file for each disk cataloged, one file per disk
FILE NAME: directory..NN where NN is the two characater disk ID
FILE TYPE: Sequential READ/WRITE file
File Structure:

Record #	Type of record	Purpose / Contents
Record 1:	String - 2 characters	Disk ID
Record 2:	Numeric	Blocks Free on disk
Record 3:	String - 16 characters	Disk Name
Record 4:	Numeric	File Count / number of files on disk
Record 5:	String - 6 characters	Date cataloged (yymmdd)

The rest of the file repeats of a group of 3 records for each disk file:

First :	String - 16 characters	File Name
Second :	Integer	File Type (128,129,130,131,132)
Third :	Integer	Number of Blocks in file

DISK ID FILE

FILE USE : File of all disk ID's in catalog
FILE NAME: disk'ids.data
FILE TYPE: Sequential READ/WRITE file
File Structure:
One record after another. Each a two character string, representing a disk ID in use.

MASTER FILE
FILE USE : File of all file names, file types, and disk ids of all files
FILE NAME: allfiles.data
FILE TYPE: Sequential READ/WRITE file
File Structure:
One record after another. Each a 19 character string. The first 16 characters are the file name, the second two it's disk's ID, and the last character the file type (p,s,r,u).

EXPLANATION OF FILE SYSTEM

Each disk cataloged into the DMS will have its own file on the MASTER DIRECTORY DISK. This makes it easy to add, delete, or change a cataloged disk directory. It makes it easy to print (or just read) any disk's directory at any time.

A separate file of Disk ID's in use makes it easy to list all the disk ID's currently cataloged. This file also makes it easy to quickly know what all the directory file names are, since each file name starts the same (directory..) and merely tacks on the ID for a file name. This makes printing a summary of all disks in the system much easier.

But to have quick access of all the file names, they should all be in one large file. This file is created by combining the file names from all the directories. To make access easiest, the file name and its disk ID are concatenated into one string record. Plus the file type is converted to a letter and tacked onto the end. Thus scanning the file names only involves reading one record per file.

But, each time a disk is added, deleted, or updated, the MASTER FILE must also be recreated.

We have worked with another DMS system that updated it's MASTER FILE every time a disk was cataloged. As the file got larger, the system got slower. Eventually it became VERY SLOW. To avoid wasting your time by constantly recreating the MASTER FILE, we allow you to quickly catalog all the disks, then just create the MASTER FILE once.

But how can you remember when you should create a new MASTER FILE? We made our DMS smart. Any time you add, delete, or update disk, a special file on the DOS MENU PROGRAM DISK is written. This file contains only one record, (TRUE or FALSE) indicating whether or not the DMS was updated since the last MASTER FILE was created. The file is written on the PROGRAM disk rather than the MASTER DISK since the MENU program will need to read this record EVERY time it is run. And since the MENU program disk is already in the drive (it just was loaded) that is the

best place to have the file.

Information on this special file:

FILE USE : File specifying whether a disk update has been performed
FILE NAME: allfiles.status
FILE TYPE: Sequential READ/WRITE file
File Structure:
Only one record, a numeric record representing TRUE or FALSE.

Now, on with business. We can procede with a clear path now that our files are defined.

```
=================
PROGRAM: VIEW'DIR
=================
```

PRELIMINARY WORK

Since our Disk Management System is based on the file names in each disk's directory, lets start off by constructing a program to simply read and display a disk's directory. Let's name this program VIEW'DIR.

We already have the introduction finished (remember INTRO). Most of the modules we write for use by VIEW'DIR will also be used by other programs in this system. Note how we construct the modules so they can be "portable" from one program to another without change.

Let's start off easy. How about a module that simply clears the screen:

```
------------
MODULE: PAGE
------------
```

We are naming the module PAGE so it will be compatible with the COMAL KERNAL extensions. The COMAL KERNAL identifies the keyword PAGE to be used to clear the screen, or to issue a form feed if a SELECT OUTPUT "LP:" is in effect. To clear the screen you merely issue a PRINT CHR$(147), statement. That is so easy you may wonder why to even make it a procedure. Mainly to make your program readable. It is nicer to see the word PAGE in a program listing. You know what is happening. If you see PRINT CHR$(147), you may not immediately know. In fact, an APPLE COMAL user wouldn't know at all.

Before typing in the procedure, clear out any old program lines:

```
NEW
```

Now, start up COMALs automatic line numbering:

```
AUTO
```

Now enter these lines:

```
PROC PAGE CLOSED
 PRINT CHR$(147), // CLEAR SCREEN
ENDPROC
```

Just hit return an extra time to stop the AUTO Line numbers. Then:

```
LIST

0010 PROC PAGE CLOSED
0020  PRINT CHR$(147), // CLEAR SCREEN
0030 ENDPROC
```

Adding the word CLOSED to the end of a procedure HEADER makes the procedure closed. All variables used in it will be considered LOCAL to the procedure and unknow to the outside program. Likewise, any variables in the outside program are unknown to the procedure. When designing modules to be used from program to program without change, you should strive to make them CLOSED if at all possible. This will remove the fear of varible conflicts. I know this procedure doesn't even have any variables used. But it is CLOSED as a HABIT.

Now to clear the screen we just enter:

```
PAGE
```

What? The screen didn't clear? Did you type in the lines right?

Oh, I forgot to tell you. You can call a procedure from direct mode any time AFTER the program is RUN. This is because COMAL compiles the program first before actually running it (know as a run time compiler). Thus once RUN the COMAL system knows about EVERY procedure and function in the program. However, if you add, delete, or change any program line COMAL forgets all about the procedures and you must issue a RUN command again. Try it:

Wait !!! Did you save it first: SAVE "PAGE1"

Now try it:

```
RUN
```

Still nothing! That's because the procedure hasn't been called yet. Try:

```
PAGE
```

Aha. Now the screen cleared. List the procedure:

```
LIST

0010 PROC PAGE CLOSED
0020  PRINT CHR$(147), // CLEAR SCREEN
0030 ENDPROC PAGE
```

Notice that COMAL has put the proc name after the ENDPROC for you. COMAL is always doing nice things like that for you.

It looks like the procedure is finished. So lets renumber it into the 9000's and LIST it to disk for use later:

```
RENUM 9000
LIST "PAGE.L"
```

MODULE: INTRO

Now, I hate to tell you this, but we should now go back and fix our INTRO procedure. It included a PRINT CHR$(147), statement. We can now change it into a simple PAGE. First clear out any program lines in the computer, then retrieve INTRO:

```
NEW
ENTER "INTRO.L"

LIST

9000 PROC INTRO(TITLE$)
9010  PRINT CHR$(147),
9020  PRINT CHR$(14), // lower case mode
9030  BACKGROUND 0
9040  BORDER 12
9050  PENCOLOR 12
9060  PRINT TITLE$
9070 ENDPROC INTRO
```

Now cursor up to line 9010 and change it to PAGE. It now looks like:

```
9010 PAGE
```

While we are fixing this procedure, let's make it CLOSED. Add the word to the end of the HEADER. Just cursor up to line 9000 and cursor over to after the) and add CLOSED. It now looks like:

```
9000 PROC INTRO(TITLE$) CLOSED
```

We may as well make another change while we are at it. Lets CENTER the title on the top of the screen. And lets make it in REVERSE FIELD. And draw a line under it for emphasis. Just type in the following lines.

In order to center the title, we need to make some calculations. The length of the line is 40 characters. First we need to know how long the title is:

```
9002 LENGTH=LEN(TITLE$)
```

Then we merely calculate how many spaces to print before and after the title in order to center it (yes we must print spaces after it too, since the top line is supposed to be all in reverse field. So we next find out the LEFT and RIGHT side spaces:

```
9004 LEFT=(40-LENGTH) DIV 2
```

Ooops! We can't use the word LEFT as a variable since it is a keyword (part of the turtle graphics system). Easy to correct. Just insert an apostrophe after it:

```
9004 LEFT'=(40-LENGTH) DIV 2
```

We are using DIV to divide by two with an integer answer. Next the right side (remember to add the ' after RIGHT since it also is a turtle keyword):

```
9006 RIGHT'=40-LEFT'-LENGTH
```

The calculations are done. Merely print the title:

```
9052 PRINT CHR$(18), // REVERSE FIELD ON
9054 FOR X=1 TO LEFT' DO PRINT " ",
9056 PRINT TITLE$,
9058 FOR X=1 TO RIGHT' DO PRINT " ",
9060 FOR X=1 TO 40 DO PRINT CHR$(162),
```

```
9062 PRINT CHR$(146), // REVERSE OFF
```

That's it. CHR$(162) is the nice graphic underline character. Notice how we replaced the original line 9060 by simply typing in a new line 9060. Now renumber it and list it:

```
RENUM 9000
LIST

9000 PROC INTRO(TITLE$) CLOSED
9010  LENGTH:=LEN(TITLE$)
9020  LEFT':=(40-LENGTH) DIV 2
9030  RIGHT':=40-LEFT'-LENGTH
9040  PAGE
9050  PRINT CHR$(14), // lower case mode
9060  BACKGROUND 0
9070  BORDER 12
9080  PENCOLOR 12
9090  PRINT CHR$(18), // REVERSE FIELD ON
9100  FOR X:=1 TO LEFT' DO PRINT " ",
9110  PRINT TITLE$,
9120  FOR X:=1 TO RIGHT' DO PRINT " ",
9130  FOR X:=1 TO 40 DO PRINT CHR$(162),
9140  PRINT CHR$(146), // REVERSE OFF
9150 ENDPROC INTRO
```

Notice that your equal signs were converted into := by COMAL. COMAL makes a distinction between = meaning a comparison and the := meaning an assignment. All the = we typed in were assignments, so COMAL added the preceding colon for us. COMAL is a real buddy.

Now, test it out. Save it first:

```
SAVE "INTRO3"
RUN
INTRO("AM I CENTERED")
```

Ooops! Now we are dependent on our PAGE procedure. This easy to correct. We merely merge our PAGE procedure into our program. But remember, all our procedures LISTED to disk have line numbers in the 9000's. So first renumber INTRO like a program, then merge in PAGE, and test it:

```
RENUM
ENTER "PAGE.L"
RUN
INTRO("AM I CENTERED")
```

It worked. So now we can LIST our new final INTRO proc onto disk. But now we have more than just INTRO, we also have PAGE as part of the program. No problem. First renumber beginning at 9000, then LIST the program, noting the start and end lines of INTRO:

```
RENUM 9000,2
LIST

9000 PROC INTRO(TITLE$) CLOSED
9002  LENGTH:=LEN(TITLE$)
9004  LEFT':=(40-LENGTH) DIV 2
9006  RIGHT':=40-LEFT'-LENGTH
9008  PAGE
9010  PRINT CHR$(14), // lower case mode
9012  BACKGROUND 0
9014  BORDER 12
9016  PENCOLOR 12
9018  PRINT CHR$(18), // REVERSE FIELD ON
9020  FOR X:=1 TO LEFT' DO PRINT " ",
9022  PRINT TITLE$,
9024  FOR X:=1 TO RIGHT' DO PRINT " ",
9026  FOR X:=1 TO 40 DO PRINT CHR$(162),
9028  PRINT CHR$(146), // REVERSE OFF
9030 ENDPROC INTRO
9032 PROC PAGE CLOSED
9034  PRINT CHR$(147), // CLEAR SCREEN
9036 ENDPROC PAGE
```

When several procedures, or a whole program is in the memory, there may not be enough line numbers available when starting at 9000 and numbering by 10's (9999 is the largest line number COMAL accepts). So just specify the interval between lines as 2, and have plenty of room. Now we also can see that INTRO starts at line 9000 and ends on line 9030. So to list only INTRO to disk, we merely specify those lines:

```
DELETE "0:INTRO.L"
LIST 9000-9030,"INTRO.L"
```

Notice we erased the old file first.

INTERMISSION

Continuing on with our VIEW'DIR program. Next we construct the modules used to read a disk's directory. We could have made it all one big module, but smaller is better, so we broke it up into several modules. The main procedure is READ'DIR2. It calls DIR'HEADER, NEXT'FILE, GET'TYPE#, GET'NAME, GET'BLOCKS#, and CHECK'DIR'CYCLE to accomplish its task. Any time a directory is to be read, we will use these sub modules, so let's refer to them collectively as READ'DIR'PART2. They, in turn, depend upon the four DISK'GET modules, which were written several years ago, allowing COMAL to access data on the disk (including the directory) one byte at a time.

First, get the four DISK'GET modules. They are on the original C64 COMAL SYSTEM DISK as well as on the CAPTAIN COMAL disk (now labeled DOS MENU PROGRAM DISK) lets get them from there. Insert that disk then:

```
NEW
ENTER "DISK'GET.L"
```

Put the disk you are using while working on these exercises back in the drive. Then:

```
LIST "DISK'GET.L"
```

Since the DISK GET routines rely on a machine code program, I won't explain how it works. To use them merely call DISK'GET'INIT first (before any other DISK'GET routine). Then call DISK'GET, DISK'GET'STRING, or DISK'GET'SKIP with the proper parameters.

```
LIST

9000 //
9010 // 4 DISK GET ROUTINES FOLLOW: (code is for a C64)
9020 //
9030 FUNC DISK'GET(FILE'NUM,REF FILE'END) CLOSED
9040  POKE 2026,FILE'NUM // PET/CBM use 636 instead of 2026
9050  SYS 2025 // PET/CBM use 635 instead of 2025
9060  FILE'END:=PEEK(144) // PET/CBM use 150 instead of 144
9070  RETURN PEEK(2024) //VALUE OF CHARACTER - PET/CBM use 634
9080 ENDFUNC DISK'GET
9090 //
9100 PROC DISK'GET'INIT CLOSED
```

```
9110  FOR LOC#:=2024 TO 2039 DO // PET/CBM use 634 TO 649
9120   READ V
9130   POKE LOC#,V
9140  ENDFOR LOC#
9150 DATA 0,162,0,32,198,255,32,207
9160 DATA 255,141,232,7,32,204,255,96 // C64 use this line
9165 //DATA 255,141,122,2,32,204,255,96 // PET/CBM use this line
9170 ENDPROC DISK'GET'INIT
9180 //
9190 PROC DISK'GET'SKIP(COUNT,FILE'NUM,REF FILE'END) CLOSED
9200  FOR X#:=1 TO COUNT DO Y:=DISK'GET(FILE'NUM,FILE'END)
9210 ENDPROC DISK'GET'SKIP
9220 //
9230 PROC DISK'GET'STRING(REF ITEM$,COUNT,FILE'NUM,REF FILE'END) CLOSED
9240  ITEM$:=""
9250  FOR X#:=1 TO COUNT DO ITEM$(X#):=CHR$(DISK'GET(FILE'NUM,FILE'END))
9260 ENDPROC DISK'GET'STRING
9270 //
9280 // END OF DISK GET ROUTINES
9290 //
```

Now, I know what you are thinking. We are almost done with this book, and haven't even done anything yet, except clear the screen. But you now have a good foundation to build on. We can now progress faster, and the final product will be sturdy. So, the rambling and repetition will decrease now. I know you can handle it. As the modules are entered, I will sprinkle in comments and explanations to help you. The AUTO line numbering will continue of course.

NOTE: comments for your benefit are sometimes included to the right of a program line. You should not type them in. They are just explaining the flow of the routine.

MODULE: READ'DIR2

PURPOSE: Read a disk's directory and optionally print it.
MODULES REQUIRED: four DISK'GET modules, six READ'DIR'PART2 modules

Now let's construct our READ'DIR2 module (it ends with a 2 because the main READ'DIR module puts the names into an array, while in this program it is not necessary). It will be sharing a bit of information with the main program, so we shouldn't make it CLOSED (all the modules that it calls will be closed though). We will pass three parameters into this module: the file number to use, the drive to use ("0" or "1"), and

whether or not to print the directory as it is read (TRUE or FALSE). These parameters help to make the procedure portable. The file number can be changed at will to adapt to any program (two files open cannot have the same file number). If a 4040 dual drive is used, the procedure can read either drive "0" or drive "1" (passed as a string to make it easier on us). And allowing the procedure the option to print the directory or not allows us to use the procedue to merely READ directory information without printing anything on the screen. All these flexible options will be available WITHOUT CHANGING the procedure. It will all be done with parameters.

```
NEW
AUTO

0010 PROC READ'DIR2(D'FILE,DRIVE$,SHOW) // NOT CLOSED
0020  RESTORE
0030  DISK'GET'INIT
```

DISK'GET'INIT reads some DATA statements. The RESTORE allows us to call it several times (to read different directories) and COMAL will reset the DATA pointer back to the beginning each time. Make sure no other DATA statements are ahead of the DISK'GET'INIT routine.

```
0040  PASS "I"+DRIVE$          initialize the drive
0050  FILE'END:=FALSE          needed when using DISK'GET
0060  OPEN FILE D'FILE,"$"+DRIVE$,READ     open directory
0070  DIR'HEADER(D'FILE,FILE'END,DISK'NAME$,DISK'ID$)
```

Here we call another procedure named DIR'HEADER. All the parameters except D'FILE are used in reference by DIR'HEADER. Thus it returns values for these variables back to the calling statement (line 0070 in this case).

```
0080  IF SHOW THEN
0090   PRINT "DISK:";DISK'NAME$;"ID:";DISK'ID$
0100   PRINT
0110   PRINT "NUM FILE NAME        TYP BLOCKS"
0120   PRINT "--- ---------------- --- ------"
0130  ENDIF
0140  BC:=0; BLOCKS'FREE:=664; FILE'COUNT:=0     initialize variables
0150  REPEAT
0160   NEXT'FILE(D'FILE,FILE'END,F'NAME$,F'TYPE#,F'BLOCKS#,BC)
```

Again all the parameters except D'FILE are used in reference. Values are assigned to them by NEXT'FILE.

```
0170   IF F'TYPE#<>128 THEN                not a deleted file type
0180    BLOCKS'FREE:-F'BLOCKS#
```

Notice how easy it is to subtract an amount from a variable. You could also use: BLOCKS'FREE:=BLOCKS'FREE-F'BLOCKS# but isn't it clearer the short way.

```
0190    FILE'COUNT:+1                       fast increment by 1
0200    // HERE YOU COULD PUT THE FILE NAME INTO AN ARRAY
0210   ENDIF
0220   IF SHOW THEN
0230    PRINT USING "###": FILE'COUNT;
```

It is a real joy to use PRINT USING. It right aligns numbers for you. Merely specify how many digits to allow with a # for each digit. Decimal points can also be included - great for lining up a column of dollar and cents amounts. Also note that the line ends with a semi-colon. That means print one space and stay on the same line.

```
0240    PRINT F'NAME$,TAB(22),TYPE$(F'TYPE#);
```

The TAB is used to keep columns lined up nicely. TYPE$ is a string array. It will be dimensioned (DIM TYPE$(128:132) OF 3) at the beginning of the program, and the five file types (DEL, SEQ, PRG, USR, REL) will be assigned to the appropriate array index. Indexes of 128 through 132 are used since those are the numbers used by the 1541 and 4040 to indicate file type. COMAL is really nice to allow us to start our array at 128 instead of 1.

```
0250    PRINT USING "###": F'BLOCKS#
0260   ENDIF
0270  UNTIL FILE'END                  continue until end of directory
0280  CLOSE FILE D'FILE
0290  IF SHOW THEN PRINT FILE'COUNT;"FILES AND";BLOCKS'FREE;"BLOCKS FRE"
0300 ENDPROC READ'DIR2
```

Done. Renumber it into the 9000's and LIST it to disk:

```
RENUM 9000
LIST "READ'DIR2.L"
```

MODULES SET: READ'DIR'PART2

SET OF SIX MODULES

Now, the group of modules called by READ'DIR2, which we shall refer to as READ'DIR'PART2. We will enter each module, one after another, and LIST them to disk as a unit:

```
NEW
AUTO

0010 // READ'DIR'PART2
0020 // SIX MODULES IN THIS SET
```

We will let the AUTO line numbers continue through all six modules.

MODULE: DIR'HEADER

PURPOSE: Gets the disk name and disk id
MODULES REQUIRED: four DISK'GET modules

```
0030 PROC DIR'HEADER(D'FILE,REF FILE'END,REF D'NAME$,REF D'ID$) CLOSED
```

Notice, this procedure is CLOSED. Also note that to use the parameters in reference the word REF precedes the variable name. This allows the procedure to assign values to those variables. The matching variables in the calling statement will receive the new values. You may have noticed that D'NAME$ in the HEADER was called DISK'NAME$ in the calling statement in procedure READ'DIR2. COMAL doesn't mind. As far as it is concerned D'NAME$ is now the same variable as DISK'NAME$ because it is called in REFerence.

```
0040  DISK'GET'SKIP(142,D'FILE,FILE'END)
0050  DISK'GET'STRING(D'NAME$,16,D'FILE,FILE'END)    disk name
0060  DISK'GET'SKIP(2,D'FILE,FILE'END)
0070  DISK'GET'STRING(D'ID$,2,D'FILE,FILE'END)       disk id
0080  DISK'GET'SKIP(92,D'FILE,FILE'END)
0090 ENDPROC DIR'HEADER
```

```
                    ----------------
                    MODULE: NEXT'FILE
                    ----------------

PURPOSE: Gets the next file's name, file type, and number of blocks
MODULES REQUIRED: GET'TYPE#, GET'NAME, GET'BLOCKS#, CHECK'DIR'CYCLE

0100 //

This line is used as a 'separator' between modules, making  them  easier
to find when glancing at a program listing.

0110 PROC NEXT'FILE(D'FILE,REF F'END,REF FN$,REF FT#,REF FB#,REF BC)
              type this at the end of above line:            CLOSED
0120  FT#:=GET'TYPE#(D'FILE,F'END)

GET'TYPE# is an integer function defined later. COMAL uses the #  symbol
after a variable name or function name to indicate that  it  is  integer
only.

0130  GET'NAME(D'FILE,F'END,FN$)
0140  FB#:=GET'BLOCKS#(D'FILE,F'END)

GET'BLOCKS# is an integer function defined later.

0150  CHECK'DIR'CYCLE(D'FILE,F'END,BC)
0160 ENDPROC NEXT'FILE

                    ----------------
                    MODULE: GET'TYPE#
                    ----------------

PURPOSE: Get the file type for this file from the directory
MODULES REQUIRED: four DISK'GET modules

0170 //
0180 FUNC GET'TYPE#(D'FILE,REF FILE'END) CLOSED
0190  THIS'TYPE#:=DISK'GET(D'FILE,FILE'END)
0200  DISK'GET'SKIP(2,D'FILE,FILE'END)
0210  IF THIS'TYPE#<129 OR THIS'TYPE#>132 THEN THIS'TYPE#:=128

Any file type value not equal to a  valid  type  of  file  (129-132)  is
converted to 128 (deleted or not used).

0220  RETURN THIS'TYPE#
```

Every function must return a value. A RETURN statement is used. Because this is an INTEGER function, the value returned must be an integer.

```
0230 ENDFUNC GET'TYPE#
```

MODULE: GET'NAME

PURPOSE: Gets the file name from the directory
MODULES REQUIRED: four DISK'GET modules

```
0240 //
0250 PROC GET'NAME(D'FILE,REF FILE'END,REF F'NAME$) CLOSED
0260  DISK'GET'STRING(F'NAME$,16,D'FILE,FILE'END)
0270  FOR SPACE#:=1 TO LEN(F'NAME$) DO
0280   IF ORD(F'NAME$(SPACE#))=160 THEN F'NAME$(SPACE#):=CHR$(32)
0290  ENDFOR SPACE#
```

This FOR loop checks each character of the file name. If it finds a SHIFTED SPACE (160) it changes it to a regular SPACE (32). ORD is used to get the ordinal value of a character. A substring is used to pick out one character at a time. CHR$ is the complement of ORD in that it takes an ordinal value and gives its character.

```
0300  DISK'GET'SKIP(9,D'FILE, FILE'END)
0310 ENDPROC GET'NAME
```

MODULE: GET'BLOCKS#

PURPOSE: Gets the number of blocks used by this file
MODULES REQUIRED: four DISK'GET modules

```
0320 //
0330 FUNC GET'BLOCKS#(D'FILE,REF FILE'END) CLOSED
0340  BLOCKS#:=DISK'GET(D'FILE,FILE'END)
0350  BLOCKS#:+256*DISK'GET(D'FILE,FILE'END)
0360  RETURN BLOCKS#
0370 ENDFUNC GET'BLOCKS#
```

MODULE: CHECK'DIR'CYCLE

PURPOSE: Skips 2 bytes after every file name cycle except every 8th one
MODULES REQUIRED: four DISK'GET modules

```
0380 //
0390 PROC CHECK'DIR'CYCLE(D'FILE,REF FILE'END,REF BLOCK'COUNT) CLOSED
0400  BLOCK'FLAG:=TRUE
0410  BLOCK'COUNT:+1
0420  IF BLOCK'COUNT=8 THEN
0430   BLOCK'COUNT:=0
0440   BLOCK'FLAG:=FALSE
0450  ENDIF
0460  IF BLOCK'FLAG THEN DISK'GET'SKIP(2,D'FILE,FILE'END)
0470 ENDPROC CHECK'DIR'CYCLE
```

Done. Now renumber it into the 9000's and LIST it to disk:

```
RENUM 9000,2
LIST "READ'DIR'PART2.L"
```

All the modules for our VIEW'DIR program are nearly finished now. All that is left is the ability to allow the option of having the directory printed on either the screen or the printer, and a module to CHAIN back to the main MENU.

MODULE: PRINTER

PURPOSE: Find out if the user wants the output on screen or printer

Screen or Printer? This is a common question for many programs. Let's put together a nice routine that takes care of this query. We will make it a function called PRINTER. It will return TRUE if a printer is desired (FALSE if the screen is to be used):

```
NEW
AUTO

0010 //
0020 FUNC PRINTER CLOSED
0030  PRINT "OUTPUT TO SCREEN OR PRINTER(S/P)",
```

The comma at the end of the line keeps us on the same line, right where we left off. Then we can print the whole word SCREEN or PRINTER at the end of the line, depending on which key the user hits.

```
0040  REPEAT
0050   CASE KEY$ OF
0060   WHEN "S","s"          watch for both shifted and unshifted S
0070    PRINT "SCREEN"
0080    RETURN FALSE
0090   WHEN "P","p"          watch for both shifted and unshifted P
0100    PRINT "PRINTER"
0110    RETURN TRUE
0120   OTHERWISE
0130    NULL                 do nothing
0140   ENDCASE
0150  UNTIL TRUE=FALSE       forever
0160 ENDFUNC PRINTER
```

KEY$ looks at the keyboard. If no key is pressed CHR$(0) is returned. We are only concerned with the S and P keys. So we set up a REPEAT loop to continuously process a multiple choice CASE based on whatever key (if any) is pressed. It does nothing until it sees a P or S (shifted or unshifted). Then it returns its value of TRUE or FALSE. The RETURN also ends the function.

Now, renumber the fucntion and list it to disk:

```
RENUM 9000
LIST "PRINTER.L"
```

MODULE: SCREEN

PURPOSE: Return output location to the screen

Of course, if the output from a program is ever directed to the printer, sooner or later we must return it back to the screen. This easily accomplished with: SELECT "DS:". However, sometimes the BUS (the group of lines between the printer, disk drive, and computer) is not properly reset and the next command to the disk'drive (such as DELETE "0:TESTING") is printed on the printer instead. This may only occur with certain printers or other rare circumstances. However, we have found that initializing a disk after a SELECT "DS:" will always clear the BUS. So, we have a quick module we can use to return output to the screen:

```
NEW
AUTO

0010 //
0020 PROC SCREEN CLOSED
0030  SELECT OUTPUT "DS:"
0040  PASS "I0"
0050 ENDPROC SCREEN
```

DONE. Now renumber and list to disk:

```
RENUM 9000
LIST "SCREEN.L"
```

MODULE: MENU

PURPOSE: Return control back to the MENU program

The last module needed for the program VIEW'DIR is the one that will link back to the main MENU. All the programs in the system will always CHAIN back to the MENU program. The MENU then allows a choice of what program to run next. This routine should first ask the user to put the DOS MENU PROGRAM DISK into drive 0 (the MENU program is on that disk):

```
NEW
AUTO 9000           this will provide 9000 range line numbers right off

9000 //
9010 PROC MENU CLOSED
9020  DIM C$ OF 1
```

CLOSED procedures are allowed to dimension strings for use just by the procedure (and modules that it calls).

```
9030  INPUT CHR$(18)+"INSERT DOS MENU PROGRAM DISK IN DRIVE 0:": C$;
```

Notice that an input prompt can use a string EXPESSION. CHR$(18) is REVERSE FIELD ON.

```
9040  PRINT "OK"+CHR$(146)+" LOADING MAIN MENU NOW ..."
```

CHR$(146) is REVERSE OFF.

```
9050  CHAIN "0:DOS'MENU"
```

```
9060 ENDPROC MENU
```

Done. LIST it to disk: LIST "MENU.L"

```
                    =================
                    PROGRAM: VIEW'DIR
                    =================
                   FINAL CONSTRUCTION
```

MODULES REQUIRED: INTRO, PAGE, MENU, PRINTER, SCREEN, READ'DIR2,
six READ'DIR'PART2 modules, four DISK'GET modules

Now the VIEW'DIR program construction begins. All we do now is write the short program. Then merely MERGE all the routines we need from disk:

```
NEW
AUTO

0010 //DELETE "0:VIEW'DIR1"
0020 //(C)1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:VIEW'DIR3"
```

All your programs should start with a similar three lines. It is really a kind of PROGRAM TRACKING SYSTEM. It allows you to keep track of what 'version' you are on. Remember when we started the book. We saved a file WELCOME1 and then a later version named WELCOME2. After awhile you might be up to WELCOME14. You then forget if you had a WELCOME15 or not. And you don't need 14 versions of the same thing on your disk. This simple three line system takes care of keeping only the latest two version of the program on disk- and remembers what version number you are on at all times. PLUS if you give out a copy of your program to a friend, later he can tell you what version it was.

Here is how the system works. All three lines start with // making them remarks. The third line contains the current program number (start with 3). The first line has the program number to be deleted next (keeps the disk from cluttering up with files, yet maintains 2 files on disk). The second line is for your name or copyright notice etc.

Each time you wish to save the program to disk, merely list the first 3 lines. Now increment the program number in the first and third lines (hit RETURN on each line). Next cursor back up to the first line. Type 7 spaces to erase the line number and the //. Hit RETURN and the DELETE command is executed. After the DELETE command is sent to the disk, the cursor returns at the beginning of the third line. Now merely type in another 7 spaces and hit RETURN and the program is SAVED with the

current program number.

It's all automatic. No more typing in a DELETE and SAVE statement. Plus, the two programs stay in the same locations in the disk directory (unless something ahead of them is deleted meanwhile).

See the COMAL TODAY newsletter issue #1 page 22 for addition info on this system. Now, back to our program:

```
0040 INTRO("VIEW CATALOG OF A DISK NOT ON THE MASTER")
```

Now we finally used that first module we wrote.

```
0050 DIM DUMMY$ OF 1
0060 DIM DISK'NAME$ OF 16
0070 DIM DISK'ID$ OF 2
0080 DIM FILE'INFO$ OF 30
0090 DIM F'NAME$ OF 16
0100 DIM TYPE$(128:132) OF 3          this is a string array
0110 TYPE$(128):="***"                a deleted file
0120 TYPE$(129):="SEQ"                sequential file
0130 TYPE$(130):="PRG"                program file
0140 TYPE$(140):="USR"                user file
0150 TYPE$(150):="REL"                relative file
0160 INPUT "[RVS]INSERT THE DISK TO VIEW INTO DRIVE 0:   [OFF]": DUMMY$
```

The input prompt will be printed in reverse field. [RVS] means enter the Reverse On key (CTRL 9). [OFF] means enter the Reverse Off key (CTRL 0). We really don't care what the user types, we merely wait for the RETURN key. This just gives him time to put the disk into the drive.

```
0170 IF PRINTER THEN SELECT OUTPUT "LP:"
```

See how we used the PRINTER function to choose the output location!

```
0180 READ'DIR2(2,"0",TRUE)
```

This calls our READ'DIR2 module, using file number 2, drive "0", and choice of printing the directory.

```
0190 SCREEN
```

This puts the output is back on the screen before the program ends.

```
0200 MENU
```

This RUNs the MENU program.

```
0210 //
```

Program is done. You can include an END statement here if you wish, but it is optional.

Now you just MERGE in the modules you previously wrote and stored on disk. Remember they use line numbers in the 9000's, so do a normal RENUM before each MERGE. To merge in a module, use the ENTER command.

```
RENUM
ENTER "INTRO.L"
RENUM
ENTER "PAGE.L"
RENUM
ENTER "MENU.L"
RENUM
ENTER "PRINTER.L"
RENUM
ENTER "SCREEN.L"
RENUM
ENTER "READ'DIR2.L"
RENUM
ENTER "READ'DIR'PART2.L"
RENUM
ENTER "DISK'GET.L"
RENUM
```

Your program is now complete. Save it first, then try a RUN:

AHA !!! Do you remember how to save the program? LIST the first three lines, increment the program numbers, and execute the first and third lines:

```
LIST -30

0010 //DELETE "0:VIEW'DIR1"
0020 //(C)1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:VIEW'DIR3"
```

Now cursor up to line 10 and change the 1 to a 2 - hit RETURN
Now cursor down to line 30 and change the 3 to a 4 - hit RETURN
Now cursor up to line 10, type 7 spaces - hit RETURN
The cursor ends up on line 30. Now type 7 spaces - hit RETURN

Done. Next program please (watch how much easier the next one will be).

==========
PROGRAM: UPDATE
==========
PRELIMINARY WORK

The next program we construct should be the most important program in this series: UPDATE. This is the program that catalogs all your disk's onto the MASTER DIRECTORY DISK. It shares many modules with our first program, VIEW'DIR.

Here are the modules we will use in this program (UPDATE) that you already have LISTed to disk: INTRO, PAGE, MENU (modified slightly into MENU2), SCREEN, DISK'GET, READ'DIR'PART2, and READ'DIR (modified from READ'DIR2).

First let's look at an overview of what this program is supposed to do.

It should be able to read a disk's directory and store it on the MASTER DIRECTORY DISK. We can use DISK'GET and READ'DIR2 (modified into READ'DIR) and READ'DIR'PART2 modules we already have written to do this. We only need to add a module to write a disk directory (WRITE'DIR) onto the MASTER DIRECTORY DISK.

It also should be able to know if a disk being cataloged is already in the system, and let the user have the option of skipping the disk (DISK ID problem) or update the entry (UPDATE the catalog). We will have to write this module (USED).

Of course the program will use INTRO and PAGE to start off, and MENU (modified into MENU2) to chain into the main MENU.

We also will need to add two more modules to make our program 'smart'. There will be a file on the DOS MENU PROGRAM DISK that records whether the user has a dual drive or not. Also is a file that tracks whether the system has been updated since the last time the MASTER FILE was created. We will include SET'UPDATED and DUAL'DRIVE to access those two files. The DOS'MENU program will create the file used by DUAL'DRIVE. The STARTUP program will create the file shared by SET'UPDATED. We will construct those programs after this one.

So, let's get on with it. First let's modify MENU and READ'DIR2 for use with this program.

MODULE: MENU2

(adapted from MENU)

PURPOSE: Return control back to the MENU program

```
NEW
ENTER "MENU.L"

LIST

9000 //
9010 PROC MENU CLOSED
9020  DIM C$ OF 1
9030  INPUT CHR$(18)+"INSERT DOS MENU PROGRAM DISK IN DRIVE 0:": C$;
9040  PRINT "OK"+CHR$(146)+" LOADING MAIN MENU NOW ..."
9050  CHAIN "0:DOS'MENU"
9060 ENDPROC MENU
```

We only need to make one small modification to the program. We need the UPDATE program to also update the file on the DOS MENU DISK that keeps track of whether or not the system was updated since the last MASTER FILE was created. We merely add the following line:

```
9045 IF FLAG THEN SET'UPDATED(TRUE)
```

Now when we call MENU we will pass it a parameter - FLAG, that is TRUE if we updated the system, and is FALSE if not. If the system was updated (FLAG=TRUE) then we call another procedure called SET'UPDATED, which we will write next. Since we are now using a parameter we must change the HEADER:

```
9010 PROC MENU2(FLAG) CLOSED
```

Notice we added the parameter FLAG and changed the name to MENU2 to differentiate it from MENU (later, in another program, we will even have a third kind of MENU: MENU3). Since we changed the procedure name, we must make the same name change in its matching ENDPROC:

```
9060 ENDPROC MENU2
```

Now, look at how the new MENU looks:

```
LIST
```

```
9000 //
9010 PROC MENU2(FLAG) CLOSED
9020  DIM C$ OF 1
9030  INPUT CHR$(18)+"INSERT DOS MENU PROGRAM DISK IN DRIVE 0:": C$;
9040  PRINT "OK"+CHR$(146)+" LOADING MAIN MENU NOW ..."
9045  IF FLAG THEN SET'UPDATED(TRUE)
9050  CHAIN "0:DOS'MENU"
9060 ENDPROC MENU2
```

That looks good. Just renumber it (to get rid of the 9045 odd number) and LIST it to disk:

```
RENUM 9000
LIST "MENU2.L"
```

Now let's write the SET'UPDATED module that MENU2 calls:

```
                    ------------------
                    MODULE: SET'UPDATED
                    ------------------
```

PURPOSE: If an update took place, it writes TRUE into the status file.

```
NEW
AUTO 9000

9000 //
9010 PROC SET'UPDATED(FLAG) CLOSED
9020  DELETE "0:ALLFILES.STATUS"
9030  OPEN FILE 78,"0:ALLFILES.STATUS",WRITE
9040  WRITE FILE 78: FLAG
9050  CLOSE FILE 78
9060 ENDPROC SET'UPDATED
```

See how easy it is to keep the status file up to date. Merely delete the old status file, then OPEN it as a new file, write the FLAG (TRUE in this case) and then CLOSE the file.

```
LIST "SET'UPDATED.L"
```

MODULE: READ'DIR

(Adapted from READ'DIR2)

PURPOSE: Read a disk's directory and store it into 3 arrays.
MODULES REQUIRED: four DISK'GET modules, six READ'DIR'PART2 modules

Let's update the READ'DIR2 procedure for reading a disk's directory. READ'DIR2 just read the directory and printed out at the same time. We will now add a few lines that allow us to STORE the directory in arrays. We can access this array in many ways. This program will merely use it to write the directory to the MASTER DIRECTORY DISK.

```
NEW
ENTER "READ'DIR2.L"

LIST

9000 PROC READ'DIR2(D'FILE,DRIVE$,SHOW) // NOT CLOSED
9010  RESTORE
9020  DISK'GET'INIT
9030  PASS "I"+DRIVE$
9040  FILE'END:=FALSE
9050  OPEN FILE D'FILE,"$"+DRIVE$,READ
9060  DIR'HEADER(D'FILE,FILE'END,DISK'NAME$,DISK'ID$)
9070  IF SHOW THEN
9080   PRINT "DISK:";DISK'NAME$;"ID:";DISK'ID$
9090   PRINT
9100   PRINT "NUM FILE NAME         TYP BLOCKS"
9110   PRINT "--- ---------------- --- ------"
9120  ENDIF
9130  BC:=0; BLOCKS'FREE:=664; FILE'COUNT:=0
9140  REPEAT
9150   NEXT'FILE(D'FILE,FILE'END,F'NAME$,F'TYPE#,F'BLOCKS#,BC)
9160   IF F'TYPE#<>128 THEN
9170    BLOCKS'FREE:-F'BLOCKS#
9180    FILE'COUNT:+1
9190    // HERE YOU COULD PUT THE FILE NAME INTO AN ARRAY
9200   ENDIF
9210   IF SHOW THEN
9220    PRINT USING "###": FILE'COUNT;
9230    PRINT F'NAME$,TAB(22),TYPE$(F'TYPE#);
9240    PRINT USING "###": F'BLOCKS#
9250   ENDIF
```

```
9260  UNTIL FILE'END
9270  CLOSE FILE D'FILE
9280  IF SHOW THEN PRINT FILE'COUNT;"FILES AND";BLOCKS'FREE;"BLOCKS FRE"
9290 ENDPROC READ'DIR2
```

First let's change the name to just READ'DIR:

```
EDIT 9000
```

```
9000 PROC READ'DIR2(D'FILE,DRIVE$,SHOW) // NOT CLOSED
```

Cursor up and over and delete the 2 (it should now look like):

```
9000 PROC READ'DIR(D'FILE,DRIVE$,SHOW) // NOT CLOSED
```

Now change the ENDPROC too:

```
9290 ENDPROC READ'DIR
```

Now, notice this line:

```
9190    // HERE YOU COULD PUT THE FILE NAME INTO AN ARRAY
```

That is just we are going to do now. First, we must agree on the array names. One array can store all the file names, called FILE'NAMES$. Another can store all the file types, called FILE'TYPES# (an integer array is used to save memory). The final array can store all the blocks for each file, called FILE'BLOCKS# (also an integer array).

These arrays will be DIMensioned at the start of the program. We need to know the upper limit to use for the arrays. Since only 144 files can be stored on a 1541 or 4040 disk, that seems the logical choice. Let's use a variable called MAX'FILES as the top of array indicator. The DIMs could then look like this:

```
MAX'FILES=144
DIM FILE'NAMES$(1:MAX'FILES) OF 16, FILE'TYPES#(1:MAX'FILES)
DIM FILE'BLOCKS#(1:MAX'FILES)
```

Now all we need to do is add three lines to our procedure replacing line 9190. These lines will put the current file name, type, and blocks into the proper array. We can use FILE'COUNT as the index into the array, since it is counting the files for us as we go:

```
9190 // INTO ARRAYS HERE
9192 FILE'NAMES$(FILE'COUNT):=F'NAME$
```

```
9194 FILE'BLOCKS#(FILE'COUNT):=F'BLOCKS#
9196 FILE'TYPES#(FILE'COUNT):=F'TYPE#
```

We can even add just a couple line so our program will not allow more programs than set in MAX'FILES. It is quite easy. We just add an IF ... ELSE ... ENDIF into the section: First list the line in the section we will be adding this IF structure into:

```
LIST 9160-9200

9160 IF F'TYPE#<>128 THEN
9170  BLOCKS'FREE:-F'BLOCKS#
9180  FILE'COUNT:+1
9190  // INTO ARRAYS HERE
9192  FILE'NAMES$(FILE'COUNT):=F'NAME$
9194  FILE'BLOCKS#(FILE'COUNT):=F'BLOCKS#
9196  FILE'TYPES#(FILE'COUNT):=F'TYPE#
9200 ENDIF
```

We should check if MAX'FILES is met or exceded right away (and if so print a warning message but skip array storage):

```
9162 IF FILE'COUNT>=MAX'FILES THEN
9164  PRINT "TOO MANY FILES"
9166 ELSE
```

Now the rest of the lines would be execute as usual. Finally we must add the ENDIF for the IF structure:

```
9198 ENDIF
```

Now list this segment:

```
LIST 9160-9200

9160 IF F'TYPE#<>128 THEN
9162  IF FILE'COUNT>=MAX'FILES THEN
9164   PRINT "TOO MANY FILES"
9166  ELSE
9170   BLOCKS'FREE:-F'BLOCKS#
9180   FILE'COUNT:+1
9190   // INTO ARRAYS HERE
9192   FILE'NAMES$(FILE'COUNT):=F'NAME$
9194   FILE'BLOCKS#(FILE'COUNT):=F'BLOCKS#
9196   FILE'TYPES#(FILE'COUNT):=F'TYPE#
9198  ENDIF
```

```
9200 ENDIF
```

That looks right, but ouch - the line numbers are a mess. So renumber the whole thing before we LIST it to disk:

```
RENUM 9000
LIST "READ'DIR.L"
```

MODULE: DUAL'DRIVE

PURPOSE: Check whether a dual drive is in use or not.

How about that module to check whether or not a dual drive is being used. There is only one record in the status file. That record is either a 1 (TRUE) or a 0 (FALSE). Let's make it a function that returns TRUE if a dual drive is used, FALSE if not:

```
NEW
AUTO 9000

9000 //
9010 FUNC DUAL'DRIVE CLOSED
9020  DIM S$ OF 2
9030  OPEN FILE 78,"0:DUALDRIVE.STATUS",READ
9040  S$:=STATUS$
```

If it can't find or read the status file, then return false to use only one drive.

```
9050  IF S$="00" THEN
9060   READ FILE 78: DRIVES
9070  ELSE
9080   DRIVES:=FALSE
9090  ENDIF
9100  CLOSE FILE 78
9110  RETURN DRIVES
9120 ENDFUNC DUAL'DRIVE
```

That's all that's to it. Remember, the file is created by DOS'MENU (constructed soon). LIST it to disk:

```
LIST "DUAL'DRIVE.L"
```

MODULE: WRITE'DIR

PURPOSE: Write a disk directory to the MASTER. Adds its disk ID to list

This is the module that will write a disk's directory onto the MASTER DIRECTORY DISK:

```
NEW
AUTO 9000

9000 //
9010 PROC WRITE'DIR
9020  DIR'FILE=8 // SET THE FILE NUMBER TO USE
9030  OPEN FILE DIR'FILE,MASTER$+"DISK'IDS.DATA",APPEND
```

Note that the file is opened as an APPEND type file. This means that the system will open the file that already exists, read quickly past all the data and position at its end ready to write the next record. The variable MASTER$ is set to either "0:" or "1:" depending on whether a dual drive is used or not. This happens at the start of the program (coming up soon). The next line writes the disk ID at the end of the DISK ID file.

```
9040  WRITE FILE DIR'FILE: DISK'ID$
9050  CLOSE FILE DIR'FILE
9060  DELETE MASTER$+"DIRECTORY.."+DISK'ID$
9070  OPEN FILE DIR'FILE,MASTER$+"DIRECTORY.."+DISK'ID$,WRITE
9080  WRITE FILE DIR'FILE: DISK'ID$,BLOCKS'FREE,DISK'NAME$,FILE'COUNT
                add this to end of above line:              ,DATE$
```

The first thing in each directory file is the disk summary. This will allow fast summaries of disks in other programs in the DMS series. It follows the structure previously outlined in this book.

```
9090  FOR X:=1 TO FILE'COUNT DO
9100   WRITE FILE DIR'FILE: FILE'NAME$(X),FILE'TYPE#(X),FILE'BLOCKS#(X)
9110  ENDFOR X
```

This writes the file's name, type, and blocks, for each file on the disk - the number of files is indicated by FILE'COUNT.

```
9120  CLOSE FILE DIR'FILE
9130  UPDATED'FLAG:=TRUE
```

Since we just updated the system, we set a flag, UPDATED'FLAG, to TRUE. Later, when the program is done, it will be used in the call to MENU.

```
9140 ENDPROC WRITE'DIR
```

That's it. LIST it to disk:

```
LIST "WRITE'DIR.L"
```

MODULE: USED

PURPOSE: Check if a disk ID is used. If it is, ask if update is wanted

This is the function that checks if a disk's ID is already in the list, and if it is, asks if an UPDATE is desired or not. It then will return TRUE if the disk ID is used and an update is NOT wanted (otherwise it returns FALSE):

```
NEW
AUTO 9000

9000 //
9010 FUNC USED
9020  FOUND:=FALSE
9030  ID'FILE:=9
9040  OPEN FILE ID'FILE,MASTER$+"DISK'IDS.DATA",READ
9050  WHILE NOT EOF(ID'FILE) AND NOT FOUND DO
```

Here we set up a loop that will read through the disk ID file till it reaches the end, unless the ID is found and it quits the loop right there.

```
9060   READ FILE ID'FILE: TEMP'ID$
9070   IF DISK'ID$:=TEMP'ID$ THEN FOUND:=TRUE
9080  ENDWHILE
9090  CLOSE FILE ID'FILE
9100  IF FOUND THEN
9110   INPUT DISK'ID$+" ALREADY EXISTS - UPDATE? ": REPLY$
9120   IF REPLY$="Y" OR REPLY$="y" THEN FOUND:=FALSE
```

We allow both shifted or unshifted Y as a YES reply.

```
9130  ENDIF
```

```
9140  RETURN FOUND
9150 ENDFUNC USED
```

Done. List it to disk: LIST "USED.L"

MODULE: UPDATE

PURPOSE: Cycle through the steps necessary to catalog a disk.
MODULES REQUIRED: READ'DIR, USED, WRITE'DIR

This is the module that will cycle through the steps necessary to catalog a disk:

```
NEW
AUTO 9000

9000 //
9010 PROC UPDATE
9020  PRINT "[RVS]PUT DISK TO CATALOG IN DRIVE 0 (Q=QUIT):[OFF]"
```

Hit CTRL 9 in place of [RVS] and hit CTRL 0 in place of [OFF].

```
9030  INPUT REPLY$
9040  IF REPLY$="Q" OR REPLY$="q" THEN
9050   DONE:=TRUE
9060   PRINT "DONE"
9070  ELSE
9080   DONE:=FALSE
9090   PRINT "OK"
9100   READ'DIR(2,"0",TRUE)
```

The parameters mean use file number 2, use drive 0, and show the directory as it is read.

```
9110  INPUT "[RVS]INSERT MASTER DIRECTORY DISK IN DRIVE "+MASTER$+
                                                  "[OFF]": REPLY$
```

The above is all one line. Hit CTRL 9 in place of [RVS] and CTRL 0 in place of [OFF].

```
9120   PRINT "OK"
9130   IF NOT USED THEN WRITE'DIR
9140  ENDIF
9150 ENDPROC UPDATE
```

Done. List it to disk: LIST "UPDATE.L"

```
               ===============
               PROGRAM: UPDATE
               ===============
            FINAL CONSTRUCTION

OLD MODULES REQUIRED: six READ'DIR'PART2 modules, four DISK'GET modules,
                      INTRO, PAGE
MODIFIED MODULES: READ'DIR (from READ'DIR2), MENU2 (from MENU)
NEW MODULES: SET'UPDATED, DUAL'DRIVE, UPDATE, USED, WRITE'DIR
```

We are now ready to construct the program. First let's get the main program written, then MERGE in all the modules we need.

```
NEW
AUTO

0010 //DELETE "0:UPDATE1"
0020 //(C) 1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:UPDATE3"
```

We started with our typical top three lines. These lines should be the top three on all your programs.

```
0040 INTRO("UPDATE THE MASTER DIRECTORY DISK")
0050 MAX'FILES:=144; UPDATED'FLAG:=FALSE
0060 DIM DISK'NAME$ OF 16, DISK'ID$ OF 2, F'NAME$ OF 16, REPLY$ OF 1
0070 DIM TEMP'ID$ OF 2, MASTER$ OF 2, DATE$ OF 6
0080 DIM FILE'TYPES#(1:MAX'FILES), FILE'BLOCKS#(1:MAX'FILES)
0090 DIM TYPE$(128:132) OF 3, FILE'NAMES$(1:MAX'FILES) OF 16
0100 TYPE$(128):="***"; TYPE$(129):="SEQ"; TYPE$(130):="PRG"
0110 TYPE$(131):="USR"; TYPE$(132):="REL"
0120 MASTER$:="0:"
0130 IF DUAL'DRIVE THEN MASTER$:="1:"
```

Here is where we check for dual drives. If TRUE is returned, we change the master drive to "1:".

```
0140 INPUT "ENTER TODAYS DATE (YYMMDD): ": DATE$
```

Notice, we are asking for the date in year, month, day. This allows for easy sorting by date for anyone adding extensions onto this system.

```
0150 REPEAT
```

```
0160  UPDATE
0170 UNTIL DONE
0180 CLOSE
0190 MENU2(UPDATED'FLAG)
```

That's all we need for the program. (Hit RETURN an extra time to quit AUTO numbering mode.) We now construct the rest of it from the modules already on the disk:

```
ENTER "INTRO.L"
RENUM
ENTER "PAGE.L"
RENUM
ENTER "MENU2.L"
RENUM
ENTER "SET'UPDATED.L"
RENUM
ENTER "DUAL'DRIVE.L"
RENUM
ENTER "READ'DIR.L"
RENUM
ENTER "READ'DIR'PART2.L"
RENUM
ENTER "UPDATE.L"
RENUM
ENTER "USED.L"
RENUM
ENTER "WRITE'DIR.L"
RENUM
ENTER "DISK'GET.L"
RENUM
```

Your program construction is now complete. The modules can be entered in any order. They can even come before the program itself. COMAL is very tolerant with us in this regard.

Remember to save the program BEFORE you try running it:

```
SAVE "UPDATE"
```

AHA !! Caught you. You should save the program using the top three lines. A habit you soon will have:

LIST -30

```
0010 //DELETE "0:UPDATE1"
0020 //(C) 1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:UPDATE3"
```

Now change the 1 to a 2, and the 3 to a 4, then execute the top line, then the third line.

==========
PROGRAM: DOS'MENU
==========
PRELIMINARY WORK

We may as well write the MENU program next. We have referred to it many times. It could be a simple list of options, and a quick CASE structure to CHAIN into the program picked. But this fine series of programs deserve better than that. So the menu program we will construct will have a few nice touches to it. This program's main modules, PRESENT'MENU AND PROCESS'MENU are quite long, due to the many print statements and the long CASE structure.

This program (and the next one we will do, STARTUP) contains few modules in common with the rest of the series. The only module already on the disk so far is PAGE. Others you may see again in the programs still to come.

So, what will DOS'MENU do? It should present a list of options to the user (PRESENT'MENU), and wait for the choice (PROCESS'MENU). It should have a HELP option (SEE), and check whether the system has been updated or not (UPDATED) since the last MASTER FILE was created. This program is the one that asks whether or not a dual drive is being used (DUAL'CHOICE). Other supporting modules used in the program are FILE'EXISTS (from the COMAL HANDBOOK) and CHOICES (from ORDER PROCESSING SYSTEM - not published yet). Also, the SEE procedure is taken from the C64 COMAL SYSTEM DISK and includes the procedure SHIFT'WAIT (from the COMAL HANDBOOK).

MODULE: SEE

PURPOSE: Display a text file onto the screen.
MODULES REQUIRED: PAGE, SHIFT'WAIT

We'll take the SEE procedure I wrote for the C64 SYSTEM DISK. It is

already written and is portable, so we'll use it. One thing I hope you have learned, is that it helps to REUSE modules from both your previous programs as well as other peoples (with their permission of course). I will explain the procedure for you as we go, since it involves some tricks. It is designed to read a sequential text file created by WORD PRO. The text file is created while in CBM printer mode as a DISK PRINTER file. Word Pro mixes up the square brackets, so this procedure fixes them. Also, we must delete the first CHR$(17) Word Pro inserted at the start of each line. The procedure also relies on the SHIFT'WAIT procedure (which we'll do next):

```
NEW
AUTO 9000

9000 //
9010 PROC SEE(FILE'NAME$) CLOSED
9020  DIM TEXT$ OF 80
9030  SEE'FILE:=106
9040  CLOSE FILE SEE'FILE // MAKE SURE FILE IS CLOSED
9050  OPEN FILE SEE'FILE,FILE'NAME$,READ
9060  PAGE
9070  PRINT // SKIP FIRST PRINT LINE
9080  REPEAT
9090   INPUT FILE SEE'FILE: TEXT$
9100   IF CHR$(17) IN TEXT$ THEN TEXT$(CHR$(17) IN TEXT$):=CHR$(0)
```

This gets rid of the CURSOR DOWN character Word Pro puts in.

```
9110   WHILE CHR$(219) IN TEXT$ DO TEXT$(CHR$(219) IN TEXT$):=CHR$(91)
```

This is how you can scan through a whole string, searching for any one character, and replacing it with another single character. In this case we are fixing the first square bracket ([). The next line fixes the other square bracket (]).

```
9120   WHILE CHR$(221) IN TEXT$ DO TEXT$(CHR$(221) IN TEXT$):=CHR$(93)
9130   SHIFT'WAIT // WAIT TILL SHIFT KEY IS DEPRESSED
9140   PRINT TEXT$
9150  UNTIL EOF(SEE'FILE)
```

Keep going until we hit the End Of File.

```
9160  CLOSE FILE SEE'FILE
9170  INPUT "FINISHED - HIT <RETURN> WHEN READY": TEXT$;
9180  PRINT
9190 ENDPROC SEE
```

Done. List it to disk: LIST "SEE.L"

MODULE: SHIFT'WAIT

PURPOSE: Wait until the shift key is depressed.

This is the module that waits for the shift key to be depressed. It comes from the COMAL HANDBOOK (you do have the book, don't you?) It is machine dependent. If you are using a PET/CBM computer, you must change the PEEK location from 653 to 152:

```
NEW
AUTO 9000

9000 //
9010 PROC SHIFT'WAIT CLOSED
9020  SHIFT'FLAG:=653 // PET USE 152
9030  WHILE NOT PEEK(SHIFT'FLAG) DO
9040   PRINT "PRESS SHIFT[UP]"
```

Hit the CURSOR UP key in place of [UP] in both line 9040 and 9050.

```
9050   PRINT "           [UP]"
9060  ENDWHILE
9070 ENDPROC SHIFT'WAIT
```

Done. List it to disk: LIST "SHIFT'WAIT.L"

MODULE: CHOICES

PURPOSE: Print a list of words with the first letter of each highlighted.

This module will print a list of words and highlight the first letter in each word in reverse field. This is useful to present a list of choices for a MENU. We will use it for our special HELP menu.

```
NEW
AUTO 9000

9000 //
```

```
9010 PROC CHOICES(TEXT$) CLOSED
9020  PRINT CHR$(18), // REVERSE ON
9030  MARK:=0; MAX:=LEN(TEXT$)
9040  WHILE MARK<MAX DO
9050   MARK:+1
9060   IF TEXT$(MARK)=" " THEN
```

Here we check a substring of TEXT$. If it is a space, we are still waiting for the next word, so turn REVESE OFF, print a space, and then turn REVERSE back on. If it isn't a space, print it and then turn off the reverse field.

```
9070    PRINT CHR$(146)," ",CHR$(18),
9080   ELSE
9090    PRINT TEXT$(MARK),CHR$(146),
9100   ENDIF
9110  ENDWHILE
9120  PRINT
9130 ENDPROC CHOICES
```

Done. List it to disk: LIST "CHOICES.L"

You may be interested in seeing this procedure in action. RUN it and then call it with some sample set of words:

```
RUN
CHOICES("THIS IS A TEST")

THIS IS A TEST
```

See how just the first letter of each word is highlighted!

MODULE: FILE'EXISTS

PURPOSE: Check if a specific file exists on the disk already.

Next, lets look at another module from the COMAL HANDBOOK. It checks if a file already exists on a disk and returns TRUE if it does (FALSE if not).

```
NEW
AUTO 9000

9000 //
```

```
9010 FUNC FILE'EXISTS(NAME$) CLOSED
9020  DIM S$ OF 2
9030  OPEN FILE 78,NAME$,READ
9040  S$:=STATUS$
9050  CLOSE FILE 78
9060  IF S$:="00" THEN
9070   RETURN TRUE
9080  ELSE
9090   RETURN FALSE
9100  ENDIF
9110 ENDFUNC FILE'EXISTS
```

Done. List it to disk: LIST "FILE'EXISTS.L"

MODULE: DUAL'CHOICE

PURPOSE: Ask if a dual drive is in use, and write the status file.

Now lets create the module that asks whether or not a dual drive is being used, and then writes a status file for use by the other programs.

```
NEW
AUTO 9000

9000 //
9010 PROC DUAL'CHOICE CLOSED
9020  DIM CHOICE$ OF 1
9030  INPUT "ARE YOU USING A DUAL DRIVE: ": CHOICE$;
9040  IF CHOICE$="Y" OR CHOICE$="y" THEN
9050   DUAL'STATUS:=TRUE
9060  ELSE
9070   DUAL'STATUS:=FALSE
9080  ENDIF
9090  DELETE "0:DUALDRIVE.STATUS"
9100  OPEN FILE 78,"0:DUALDRIVE.STATUS",WRITE
9110  WRITE FILE 78: DUAL'STATUS
9120  CLOSE FILE 78
9130  PRINT
9140 ENDPROC DUAL'CHOICE
```

Done. List it to disk: LIST "DUAL'CHOICE.L"

MODULE: UPDATED

PURPOSE: Check if the system has been updated by reading status file.

Now we can write the function that makes our system smart. This function reads the status file to tell if the sytem has been updated since the last MASTER FILE was created and returns TRUE if it has (FALSE if not).

```
NEW
AUTO 9000

9000 //
9010 FUNC UPDATED CLOSED
9020  DIM S$ OF 2
9030  OPEN FILE 78,"ALLFILES.STATUS",READ
9040  S$:=STATUS$
9050  IF S$="00" THEN
9060   READ FILE 78: UPDATED'FLAG
9070  ELSE
9080   PRINT "UPDATE STATUS FILE NOT FOUND"
9090   UPDATED'FLAG:=FALSE
9100  ENDIF
9110  CLOSE FILE 78
9120  RETURN UPDATED'FLAG
9130 ENDFUNC UPDATED
```

Done. List it to disk: LIST "UPDATED.L"

MODULE: PRESENT'MENU

PURPOSE: Display the options available with the system
MODULES REQUIRED: PAGE, FILE'EXISTS, DUAL'CHOICE, UPDATED

This is the first in a "pair" of modules, one to display the menu, and the other to process the choice. First, the menu display module:

```
NEW
AUTO 9000

9000 //
9010 PROC PRESENT'MENU
```

```
9020  PAGE
9030  DONE:=FALSE
9040  PRINT "  [RVS] [OFF]      CAPTAIN COMAL PRESENTS:"
```

In order to have a fancy looking menu, we are using the REVERSE ON and REVERSE OFF quite a bit. Use CTRL 9 in place of each [RVS] and CTRL 0 in place of each [OFF].

```
9050  PRINT "  [RVS] [OFF]"
9060  PRINT "[RVS]     DISK ORGANIZATION SYSTEM MENU     [OFF]",
```

Note: it is exactly 40 characters in between the [RVS] and the [OFF].

```
9070  PRINT "  [RVS] [OFF]"
9080  PRINT "  [RVS] [OFF]"
9090  PRINT "A [RVS]-[OFF] ALLOCATE FIRST TIME ONLY FILES"
9100  PRINT "C [RVS]-[OFF] COMPARE TWO DIRECTORIES"
9110  PRINT "D [RVS]-[OFF] DELETE DISK FROM THE MASTER CATALOG"
9120  PRINT "F [RVS]-[OFF] FIND A FILE IN THE MASTER CATALOG"
9130  PRINT "H [RVS]-[OFF] HELP"
9140  PRINT "I [RVS]-[OFF] ID'S IN USE CHART OR LIST"
9150  PRINT "M [RVS]-[OFF] MAKE MASTER FILE OF CATALOGED DISKS"
9160  PRINT "N [RVS]-[OFF] NUMBER OF DRIVES (SINGLE OR DUAL)"
9170  PRINT "P [RVS]-[OFF] PRINT DIRECTORY (MULTI-FORMATS)"
9180  PRINT "S [RVS]-[OFF] SUMMARY OF DISKS IN THE MASTER"
9190  PRINT "U [RVS]-[OFF] UPDATE MASTER CATALOG"
9200  PRINT "V [RVS]-[OFF] VIEW AN UNCATALOGED DISK DIRECTORY"
9210  PRINT "  [RVS] [OFF]"
9220  PRINT "Q [RVS]-[OFF] QUIT"
9230  PRINT "  [RVS] [OFF]"
9240  PRINT "  [RVS] [OFF]";
9250  IF FILE'EXISTS("0:DUALDRIVE.STATUS") THEN
9260   PRINT
9270  ELSE
9280   DUAL'CHOICE
9290  ENDIF
9300  PRINT "  [RVS] [OFF]"
9310  PRINT "  [RVS] [OFF]";
9320  IF UPDATED THEN PRINT "CHOOSE OPTON M SOON.";
9330  PRINT
9340  PRINT "  [RVS] [OFF]"
9350  PRINT "  [RVS] [OFF][UP][UP]"
```

The above line prints on the last screen line, and ends with two cursor ups.

```
9360 ENDPROC PRESENT'MENU
```

Done. List it to disk: LIST "PRESENT'MENU.L"

MODULE: PROCESS'MENU

PURPOSE: Find out what option the user wants next.
MODULES REQUIRED: CHOICES, SEE

This module asks the user to choose one of the options and ignore an incorrect response. If HELP is requested, a submenu is presented.

```
NEW
AUTO 9000

9000 //
9010 PROC PROCESS'MENU
9020  INPUT "  [RVS] [OFF] YOUR CHOICE >": CHOICE$;
9030  CASE CHOICE$ OF
9040  WHEN "A","a"
9050   PRINT "FIRST TIME ALLOCATE"
9060   PRINT "THIS WILL ZERO OUT / CREATE FILES"
9070   INPUT "CONTINUE (Y/N):": CHOICE$
9080   IF CHOICE$="Y" OR CHOICE$="y" THEN CHAIN "STARTUP"
9090  WHEN "C","c"
9100   PRINT "COMPARE DIRECTORIES"
9110   CHAIN "COMPARE'DIR"
9120  WHEN "D","d"
9130   PRINT "DELETE DISK ENTRY"
9140   CHAIN "DELETE'DIR"
9150  WHEN "F","f"
9160   PRINT "FIND A FILE"
9170   CHAIN "FIND'FILE"
9180  WHEN "H","h","?","/"
```

For requesting HELP, we are also watching for the question mark key, shifted (?) or unshifted (/).

```
9190   PRINT "HELP - INSTRUCTIONS"
9200   CHOICES("GRAPHICS SPRITES COMAL INSTRUCTIONS ?")
9210   REPEAT
9220    OK:=TRUE
9230    CASE KEY$ OF
9240    WHEN "G","g"
```

```
9250      SEE("HELP-GRAPHICS")
9260     WHEN "S","s"
9270      SEE("HELP-SPRITES")
9280     WHEN "C","c"
9290      SEE("HELP-COMAL")
9300     WHEN "I","i","H","h","?","/"
9310      SEE("HELP-INSTRUCTION")
9320     OTHERWISE
9330      OK:=FALSE
9340     ENDCASE
9350    UNTIL OK
9360   WHEN "I","i"
9370    PRINT "ID LIST OR CHART"
9380    CHAIN "PRINT'IDS"
9390   WHEN "M","m"
9400    PRINT "MAKE NEW MASTER FILE"
9410    CHAIN "MASTER'MAKER"
9420   WHEN "N","n"
9430    PRINT "NUMBER OF DRIVES"
9440    DUAL'CHOICE
9450   WHEN "P","p"
9460    PRINT "PRINT'DIRECTORIES"
9470    CHAIN "PRINT'DIR"
9480   WHEN "Q","q"
9490    PRINT "QUIT THE SYSTEM"
9500    DONE:=TRUE
9510   WHEN "S","s"
9520    PRINT "SUMMARY OF DISKS"
9530    CHAIN "DISK'SUMMARY"
9540   WHEN "U","u"
9550    PRINT "UPDATE MASTER CATALOG"
9560    CHAIN "UPDATE"
9570   WHEN "V","v"
9580    PRINT "VIEW UNCATALOGED CAT"
9590    CHAIN "VIEW'DIR"
9600   OTHERWISE
9610    NULL
9620   ENDCASE
9630 ENDPROC PROCESS'MENU
```

That's it. LIST it to disk:

```
LIST "PROCESS'MENU.L"
```

```
================
PROGRAM: DOS'MENU
================
FINAL CONSTRUCTION

OLD MODULES REQUIRED: PAGE
NEW MODULES REQUIRED: PRESENT'MENU, PROCESS'MENU, UPDATED, DUAL'CHOICE,
                      FILE'EXISTS, CHOICES, SEE, SHIFT'WAIT
```

Now we're ready to construct our DOS'MENU program.

But, wait. Did you notice how a program is CHAINed:

```
CHAIN "VIEW'DIR"
```

That gives us a problem. We save our programs with their version number after the name. Thus VIEW'DIR would not be found (it is on disk as VIEW'DIR4 or something). Yes, this is true. But, we are only in the construction stage. The diskette we are using now can be referred to as a WORKING DISK. Once a program is tested and works, save it as usual, but save it an additional time on a final diskette without the program version number attached (but keep the top three lines in the program of course).

Now lets start with the main program:

```
NEW
AUTO

0010 //DELETE "0:DOS'MENU1"
0020 //(C)1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:DOS'MENU3"
0040 DIM CHOICE$ OF 1
0050 DIM TEXT$ OF 40
0060 REPEAT
0070  PRESENT'MENU
0080  PROCESS'MENU
0090 UNTIL DONE
0100 //
```

You may make line 100 an END statement if you wish.

That is it. A very simple program. The rest, of course, is the modules already on disk. Finish the program construction:

```
ENTER "PRESENT'MENU.L"
RENUM
ENTER "PROCESS'MENU.L"
RENUM
ENTER "UPDATED.L"
RENUM
ENTER "DUAL'CHOICE.L"
RENUM
ENTER "FILE'EXISTS.L"
RENUM
ENTER "PAGE.L"
RENUM
ENTER "CHOICES.L"
RENUM
ENTER "SEE.L"
RENUM
ENTER "SHIFT'WAIT.L"
RENUM
```

The program is now complete. SAVE it in the usual way, using the first three program lines.

```
================
PROGRAM: STARTUP
================
PRELIMINARY WORK
```

Next we will construct the STARTUP program. It creates files our programs depend on.

STARTUP will create a dummy disk directory file (START'DIRECTORY) so that the system is not empty. It also creates the DISK'ID file, starting it with the dummy ID of "00" (START'DISKIDS). It also creates the status file for the MASTER FILE (START'ALL'STATUS) and starts the MASTER FILE itself with one dummy record (START'ALLFILES).

One important thing STARTUP will do is FORMAT a disk for the user. All the user needs is a blank disk, and the program will format it properly for him. Procedures FORMAT'MASTER and FORMAT'IT take care of this.

The program also shares PAGE, MENU, and INTRO modules as we previously stored on disk.

MODULE: FORMAT'MASTER

PURPOSE: Asks the user if the MASTER DISK needs to be formatted.
MODULE REQUIRED: FORMAT'IT

This is the first of two modules that take care of formatting a disk.

```
NEW
AUTO 9000

9000 //
9010 PROC FORMAT'MASTER CLOSED
9020  PRINT "TO PROPERLY USE THIS SYSTEM YOU SHOULD"
9030  PRINT "HAVE A MASTER DIRECTORY DISK. THIS DISK"
9040  PRINT "WILL BE USED TO STORE THE DIRECTORIES"
9050  PRINT "OF ALL YOUR OTHER DISKS."
9060  PRINT
9070  PRINT "IT IS BEST IF YOU HAVE A NEW BLANK DISK"
9080  PRINT "TO USE. THE SYSTEM CAN THE FORMAT IT"
9090  PRINT "FOR YOU AS THE MASTER DISK."
9100  PRINT
9110  PRINT "IF YOU ALREADY HAVE A FORMATTED MASTER"
9120  PRINT "DISK, YOU MAY REPLY NO TO THE NEXT"
9130  PRINT "QUESTION."
9140  PRINT
9150  PRINT "WOULD YOU LIKE THE SYSTEM TO FORMAT A"
9160  PRINT "MASTER DISK FOR YOU (Y/N): "
9170  REPEAT
9180   DISK'OK:=TRUE
9190   CASE KEY$ OF
9200   WHEN "Y","y"
9210    FORMAT'IT
9220   WHEN "N","n"
9230    NULL
9240   OTHERWISE
9250    DISK'OK:=FALSE
9260   ENDCASE
9270  UNTIL DISK'OK
9280 ENDPROC FORMAT'MASTER
```

Done. List it to disk: LIST "FORMAT'MASTER.L"

```
-----------------
MODULE: FORMAT'IT
-----------------
```

PURPOSE: Format a MASTER DISK after verifying that this is really wanted

Now we'll write the procedure that actually formats the disk. Since this will completely erase the disk, we must make sure this is what the user wants (give a chance to change his mind).

```
NEW
AUTO 9000

9000 //
9010 PROC FORMAT'IT CLOSED
9020  DIM S$ OF 2, ID$ OF 2
9030  REPEAT
9040   S$:="00"
9050   PRINT
9060   PRINT ">>>> FORMAT A MASTER DISK <<<<"
9070   PRINT
9080   PRINT "ENTER Q FOR QUIT TO SKIP THIS"
9090   PRINT "OR ENTER F TO FORMAT THE DISK"
9100   PRINT CHR$(18)+"INSERT DISK TO BE FORMATTED INTO DRIVE 0"+
          add this to the above line:                    CHR$(146),
```

Notice that we require an F to format, not just the RETURN key. This is to help verify that it is not a mistake. Also, you may recall that CHR$(18) is REVERSE ON and that CHR$(146) is REVERSE OFF.

```
9110   REPEAT
9120    FORMAT'OK:=TRUE
9130    CASE KEY$ OF
9140    WHEN "F","f"
9150     INPUT "ENTER 2 CHARACTER ID FOR MASTER: ": ID$
9160     ID$:=ID$+"00"
9170     PASS "N0:MASTER DISK,"+ID$(1:2)
```

The above two lines make sure that the ID is two characters. If less than two characters are entered, it merely adds two zero characters to the reply, then uses the first two characters of ID$. Yes, I know that ID$ is only dimensioned to 2. But this way, it is clear exactly what we are doing.

```
9180     PRINT ">>>> NOW FORMATTING DISK - PLEASE WAIT",
```

```
9190      S$:=STATUS$
```

By requesting the STATUS right away, the program must wait for the previous disk operation to end. This is an easy way to have the program pause until the formatting is done. In addition, we can make sure that the status was "00" for no error. If not, we report an error and try again.

```
9200      PRINT
9210      IF S$<>"00" THEN PRINT CHR$(18)+">>> DISK ERROR <<<"
9220     WHEN "Q","q"
9230      NULL
9240     OTHERWISE
9250      FORMAT'OK:=FALSE
9260     ENDCASE
9270    UNTIL FORMAT'OK
9280   UNTIL S$="00"
9290 ENDPROC FORMAT'IT
```

Done. List it to disk: LIST "FORMAT'IT.L"

MODULE: START'ALLFILES

PURPOSE: Create the MASTER FILE.

Now lets write our four file creating modules. Start with the one to create the MASTER FILE:

```
NEW
AUTO 9000

9000 //
9010 PROC START'ALLFILES
9020  ALL'FILE:=5
9030  DELETE MASTER$+"ALLFILES.DATA"
9040  OPEN FILE ALL'FILE,MASTER$+"ALLFILES.DATA",WRITE
9050  FILE'NAME$:="DUMMY           00P"
9060  WRITE FILE ALL'FILE: FILE'NAME$
9070  CLOSE FILE ALL'FILE
9080 ENDPROC START'ALLFILES
```

Done. List it to disk: LIST "START'ALLFILES.L"

MODULE: START'ALL'STATUS

PURPOSE: Create status file that tells if the system has been updated.

Next, let's do the module to create the status file specifying whether the system has been updated or not. Since at start up time, it has not been updated, we set it to FALSE to start with.

```
NEW
AUTO 9000

9000 //
9010 PROC START'ALL'STATUS
9020  ALL'FILE:=5
9030  DELETE "0:ALLFILES.STATUS"
9040  OPEN FILE ALL'FILE,"0:ALLFILES.STATUS",WRITE
9050  UPDATED:=FALSE
9060  WRITE FILE ALL'FILE: UPDATED
9070  CLOSE FILE ALL'FILE
9080 ENDPROC START'ALL'STATUS

Done. List to disk: LIST "START'ALL'STATUS.L"
```

MODULE: START'DIRECTORY

PURPOSE: Create a dummy directory file.

Next, the module that will create a dummy directory file, so that the system is not empty.

```
NEW
AUTO 9000

9000 //
9010 PROC START'DIRECTORY
9020  DIR'FILE:=8
9030  DELETE MASTER$+"DIRECTORY..??"
```

This line deletes ALL files that start with DIRECTORY.. and have two other characters after that (all directory files in this system meet that criteria).

```
9040  OPEN FILE DIR'FILE,MASTER$+"DIRECTORY..00",WRITE
9050  DISK'NAME$:="DUMMY"
9060  DATE$:="DUMMY"
9070  NUM:=1
9080  FILE'NAME$:="DUMMY"
9090  WRITE FILE DIR'FILE: DISK'ID$,NUM,DISK'NAME$,NUM,DATE$
9100  INTEGER#:=129
9110  WRITE FILE DIR'FILE: FILE'NAME$,INTEGER#,INTEGER#
9120  CLOSE FILE DIR'FILE
9130 ENDPROC START'DIRECTORY
```

Done. List it to disk: LIST "START'DIRECTORY.L"

MODULE: START'DISKIDS

PURPOSE: Create a dummy disk id file.

This final module creates the disk ID file. It starts with dummy id 00.

```
NEW
AUTO 9000

9000 //
9010 PROC START'DISKIDS
9020  ID'FILE:=9
9030  DELETE MASTER$+"DISK'IDS.DATA"
9040  OPEN FILE ID'FILE,MASTER$+"DISK'IDS.DATA",WRITE
9050  DISK'ID$:="00"
9050  WRITE FILE ID'FILE: DISK'ID$
9060  CLOSE FILE ID'FILE
9070 ENDPROC START'DISKIDS
```

Done. List it to disk: LIST "START'DISKIDS.L"

==========

PROGRAM: STARTUP

==========

FINAL CONSTRUCTION

OLD MODULES REQUIRED: INTRO, PAGE, MENU
NEW MODULES REQUIRED: START'DISKIDS, START'DIRECTORY, START'ALL'STATUS,
START'ALLFILES, FORMAT'MASTER, FORMAT'IT

All the necessary modules are now on the disk. We can begin our program construction:

```
NEW
AUTO

0010 //DELETE "0:STARTUP1"
0020 //(C)1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:STARTUP3"
0040 INTRO("SYSTEM STARTUP - CREATE FILES")
0050 DIM DISK'NAME$ OF 16, DISK'ID$ OF 2
0060 DIM MASTER$ OF 2, REPLY$ OF 1
0070 DIM FILE'NAME$ OF 19, DATE$ OF 6
0080 MASTER$:="0:"
0090 START'ALL'STATUS
```

We call START'ALL'STATUS now because it writes its file on the DOS MENU PROGRAM DISK which is still in drive 0.

```
0100 FORMAT'MASTER
0110 INPUT CHR$(18)+"INSERT MASTER DIRECTORY DISK IN DRIVE 0:": REPLY$
0120 PRINT "OK"
0130 START'DISKIDS
0140 START'DIRECTORY
0150 START'ALLFILES
0160 MENU
```

That's the program. Now finish the construction by MERGING in the modules needed from disk:

```
ENTER "START'DISKIDS.L"
RENUM
ENTER "START'DIRECTORY.L"
RENUM
ENTER "START'ALL'STATUS.L"
RENUM
ENTER "START'ALLFILES.L"
RENUM
ENTER "INTRO.L"
RENUM
ENTER "PAGE.L"
RENUM
ENTER "MENU.L"
RENUM
ENTER "FORMAT'MASTER.L"
RENUM
```

```
ENTER "FORMAT'IT.L"
RENUM
```

Program construction is now complete. Save the program on your disk (you know how to do it).

==========
PROGRAM: MASTER'MAKER
==========
PRELIMINARY WORK

The appropriate program to construct next is the program that creates a MASTER FILE from all the directory files on the MASTER DIRECTORY DISK.

The program will first sort the DISK ID file and then add directories to the MASTER FILE in sorted order. To do this we will have READ'IDS, SORT'IDS, and WRITE'SORTED modules. And we will use the three QUICKSORT modules taken from the C64 COMAL SYSTEM DISK. We will also modify MENU2 into MENU3 (the last of the MENUs). SET'UPDATED remains the same as already on disk. We also will use INTRO, PAGE, and DUAL'DRIVE unchanged from disk. We will add a module to request the MASTER DISK be inserted in the drive (INIT). Also we will need to add modules to process the MASTER FILE (PROCESS'MASTER) and add a directory to the MASTER FILE (ADD'DIRECTORY). Let's begin.

MODULE: INIT

PURPOSE: Sets the drive to use for MASTER$ and requests the MASTER DISK
MODULE REQUIRED: DUAL'DRIVE

Let's start with the INIT module. It will be used by all the rest of the DMS programs. It checks if a dual drive is being used, then asks the user to insert the MASTER DIRECTORY DISK in the appropriate drive.

```
NEW
AUTO 9000

9000 //
9010 PROC INIT(REF MASTER$) CLOSED
```

INIT uses MASTER$ as a parameter in reference so that the main program is informed of what the value of MASTER$ is.

```
9020  DIM C$ OF 1
```

```
9030  MASTER$:="0:"
9040  IF DUAL'DRIVE THEN MASTER$:="1:"
9050  INPUT CHR$(18)+"INSERT MASTER DIRECTOY DISK IN DRIVE "+MASTER$:C$;
9060  PRINT "OK"
9070 ENDPROC INIT
```

Done. List it to disk: LIST "INIT.L"

MODULE: MENU3

PURPOSE: Request DOS MENU PROGRAM DISK and update the status file.
MODULE REQUIRED: SET'UPDATED

We will simply change one line in MENU2 (plus change its name to MENU3) and this module is done.

```
NEW
ENTER "MENU2.L"

LIST

9000 //
9010 PROC MENU2(FLAG) CLOSED
9020  DIM C$ OF 1
9030  INPUT CHR$(18)+"INSERT DOS MENU PROGRAM DISK IN DRIVE 0:": C$;
9040  PRINT "OK"+CHR$(146)+" LOADING MAIN MENU NOW ..."
9050  IF FLAG THEN SET'UPDATED(TRUE)
9060  CHAIN "0:DOS'MENU"
9070 ENDPROC MENU
```

After the MASTER FILE is created, we have to set the status file to FALSE, that no update has taken place since the MASTER FILE was created. Thus we merely change line 9050 to:

```
9050 SET'UPDATED(FLAG)
```

Now change the name in the HEADER and the ENDPROC, then LIST to disk:

```
9010 PROC MENU3(FLAG) CLOSED
9070 ENDPRC MENU3
LIST "MENU3.L"
```

```
                              ---------------
                              MODULE: READ'IDS
                              ---------------
```

PURPOSE: Read the disk IDs from the file into an array.

```
NEW
AUTO 9000

9000 //
9010 PROC READ'IDS
9020  PRINT "READING DISK ID FILE"
9030  ID'FILE:=9
9040  OPEN FILE ID'FILE,MASTER$+"DISK'IDS.DATA",READ
9050  COUNT:=0
9060  WHILE NOT EOF(ID'FILED) AND COUNT<MAX'NUM'DISKS DO
9070   COUNT:+1
9080   READ FILE ID'FILE: ID$(COUNT)
9090  ENDWHILE
9100  CLOSE FILE ID'FILE
9110 ENDPROC READ'IDS
```

Done. List it to disk: LIST "READ'IDS.L"

```
                              ---------------
                              MODULE: SORT'IDS
                              ---------------
```

PURPOSE: Sort the IDs in the array.
MODULES REQUIRED: three QUICKSORT modules

```
NEW
AUTO 9000

9000 //
9010 PROC SORT'IDS
9020  PRINT "SORTING DISK IDS"
9030  QUICKSORT(ID$,1,COUNT,2)
```

QUICKSORT will do all the sorting for us. We merely tell it the array name (ID$), the starting record number (1), the number of records (COUNT), and the length of the string (2).

```
9040 ENDPROC SORT'IDS
```

Done. List it to disk: LIST "SORT'IDS.L"

MODULE: QUICKSORT

PURPOSE: Sorting utility taken from the COMAL SYSTEM DISK.

This set of three routines is on the disk that came with this book. It is not our place to discuss how it works. Merely use it. It is LISTed to the disk as QUICKSORT.L as you expected. This is what its three routines look like:

```
NEW
ENTER "QUICKSORT.L"

LIST

9000 //
9010 // 3 QUICKSORT ROUTINES FOLLOW:
9020 //
9030 PROC QUICKSORT(REF A$(),LEFT',RIGHT',RECLEN) CLOSED
9040  DIM PIVOT$ OF RECLEN, BUFFER$ OF RECLEN
9050  PARTITION(LEFT',RIGHT',LEFT',RIGHT') // SORT A$(LEFT':RIGHT')
9060 ENDPROC QUICKSORT
9070 //
9080 PROC PARTITION(LEFT',RIGHT',I,J)
9090  PIVOT$:=A$((LEFT'+RIGHT') DIV 2) // GET MIDDLE ELEMENT AS PIVOT
9100  REPEAT  // PERFORM SWAPPINGS
9110   WHILE PIVOT$>A$(I) DO I:+1
9120   WHILE PIVOT$<A$(J) DO J:-1
9130   IF I<=J THEN SWAP(A$(I),A$(J)); I:+1; J:-1
9140  UNTIL I>J
9150  IF LEFT'<J THEN PARTITION(LEFT',J,LEFT',J) // SORT A$(LEFT':J)
9160  IF I<RIGHT' THEN PARTITION(I,RIGHT',I,RIGHT') // SORT A$(I:RIGHT')
9170 ENDPROC PARTITION
9180 //
9190 PROC SWAP(REF A$,REF B$)
9200  BUFFER$:=A$; A$:=B$; B$:=BUFFER$
9210 ENDPROC SWAP
9220 //
9230 // END OF QUICKSORT ROUTINES
9240 //
```

```
--------------------
MODULE: WRITE'SORTED
--------------------
```

PURPOSE: Write disk ID file in sorted order. Duplicate ID's are ignored.

```
NEW
AUTO 9000

9000 //
9010 PROC WRITE'SORTED
9020  NEW'FILE:=10
9030  PRINT "WRITING NEW SORTED DISK ID FILE"
9040  DELETE MASTER$+"NEW'IDS"
9050  OPEN FILE NEW'FILE,MASTER$+"NEW'IDS",WRITE
9060  LAST'ID$:=""
```

By keeping track of what the last ID written to the file was we can ignore any duplicate IDs.

```
9070  FOR X:=1 TO COUNT DO
9080   IF ID$(X)>LAST'ID$ THEN
9090    WRITE FILE NEW'FILE: ID$(X)
9100    LAST'ID$:=ID$(X)
9110   ENDIF
9120  ENDFOR X
9130  CLOSE FILE NEW'FILE
9140  DELETE MASTER$+"OLD'IDS.DATA"
9150  PASS "R"+MASTER$+"OLD'IDS.DATA=DISK'IDS.DATA"
9160  PASS "R"+MASTER$+"DISK'IDS.DATA=NEW'IDS"
```

The above three lines get rid of the old 'backup' ID file, then rename the current ID file to be the old file, then rename the new ID file as the current file.

```
9170 ENDPROC WRITE'SORTED
```

Done. List it to disk: LIST "WRITE'SORTED.L"

```
                        ----------------------
                        MODULE: PROCESS'MASTER
                        ----------------------

PURPOSE: Add each directories files. Use disk ID file for directory IDs.
MODULE REQUIRED: ADD'DIRECTORY

NEW
AUTO 9000

9000 //
9010 PROC PROCESS'MASTER
9020  ID'FILE:=9; ALL'FILE:=3
9030  DELETE MASTER$+"ALLFILES.DATA"
9040  OPEN FILE ALL'FILE,MASTER$+"ALLFILES.DATA",WRITE
9050  OPEN FILE ID'FILE,MASTER$+"DISK'IDS.DATA",READ
9060  WHILE NOT EOF(ID'FILE) DO
9070   READ FILE ID'FILE: ADD'ID$
9080   ADD'DIRECTORY(ADD'ID$,SHOW'DIR)

The variable SHOW'DIR is set in the main program. If it is TRUE then the
file names in each directory wil be printed as they are added.

9090  ENDWHILE
9100  CLOSE FILE ID'FILE
9110  CLOSE FILE ALL'FILE
9120 ENDPROC PROCESS'MASTER

Done. List it to disk: LIST "PROCESS'MASTER.L"

                        ---------------------
                        MODULE: ADD'DIRECTORY
                        ---------------------

PURPOSE: Adds the files into the MASTER FILE. Prints them as it goes.

NEW
AUTO 9000

9000 //
9010 PROC ADD'DIRECTORY(ID$,SHOW)
9020  DIR'FILE:=8
9030  PRINT "ADDING DIRECTORY ..";ID$
9040  OPEN FILE DIR'FILE,MASTER$+"DIRECTORY.."+ID$,READ
9050  READ FILE DIR'FILE: DISK'ID$,X,DUMMY$,X,DUMMY$
```

We can just skip over the summary information stored at the beginning of the file.

```
9060  WHILE NOT EOF(DIR'FILE) DO
9070   READ FILE DIR'FILE: FILE'NAME$,FILE'TYPE#,BLOCK#
9080   FILE'NAME$(1:19):=FILE'NAME$
```

This makes sure that the string is 19 characters long. COMAL puts in spaces at the end to make it 19 characters.

```
9090   FILE'NAME$(17:18):=DISK'ID$
```

The disk ID comes right after the 16 characters reserved for the file name.

```
9100   FILE'NAME$(19):=TYPE$(FILE'TYPE#)
```

We tack on the file type as a one letter string in case future expansion to this system needs this information from the master file. We use a string array (TYPE$) for this purpose. It is set up in the beginning of the program.

```
9110   WRITE FILE ALL'FILE: FILE'NAME$
9120   IF SHOW THEN PRINT FILE'NAME$
9130  ENDWHILE
9140  CLOSE FILE DIR'FILE
9150  IF SHOW THEN PRINT "---------------------------------"
9160 ENDPROC ADD'DIRECTORY
```

Done. LIST it to disk: LIST "ADD'DIRECTORY.L"

```
                          ====================
                          PROGRAM: MASTER'MAKER
                          ====================
                           FINAL CONSTRUCTION

OLD MODULES: INTRO, PAGE, DUAL'DRIVE, SET'UPDATED
MODIFIED MODULES: MENU3 (from MENU2)
NEW MODULES: PROCESS'MASTER, ADD'DIRECTORY, INIT, READ'IDS, SORT'IDS,
              WRITE'SORTED, three QUICKSORT modules
```

All the needed modules are now on disk. We can write the program and then MERGE in the modules.

```
NEW
```

```
AUTO

0010 //DELETE "0:MASTER'MAKER1"
0020 //(C)1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:MASTER'MAKER3"
0040 INTRO("MAKE A NEW MASTER FILE OF ALL FILES")
0050 MAX'NUM'DISKS:=140; SHOW'DIR:=TRUE
0060 DIM ID$(1:MAX'NUM'DISKS) OF 2
0070 DIM LAST'ID$ OF 2
0080 DIM DISK'ID$ OF 2, FILE'NAME$ OF 19, TYPE$(129:132) OF 1
0090 DIM DUMMY$ OF 1, MASTER$ OF 2, ADD'ID$ OF 2
0100 TYPE$(129):="S"; TYPE$(130):="P"; TYPE$(131):="U"; TYPE$(132):="R"
0110 INIT(MASTER$)
0120 READ'IDS
0130 SORT'IDS
0140 WRITE'SORTED
0150 PROCESS'MASTER
0160 MENU3(FALSE)
```

That's the program. Now merely MERGE in the modules needed and the construction is complete:

```
ENTER "PROCESS'MASTER.L"
RENUM
ENTER "ADD'DIRECTORY.L"
RENUM
ENTER "INTRO.L"
RENUM
ENTER "PAGE.L"
RENUM
ENTER "INIT.L"
RENUM
ENTER "DUAL'DRIVE.L"
RENUM
ENTER "MENU3.L"
RENUM
ENTER "SET'UPDATED.L"
RENUM
ENTER "READ'IDS.L"
RENUM
ENTER "SORT'IDS.L"
RENUM
ENTER "WRITE'SORTED.L"
RENUM
ENTER "QUICKSORT.L"
RENUM
```

The program construction is now complete. Save it to disk. You know how.

==========
PROGRAM: PRINT'DIR
==========
PRELIMINARY WORK

This program will print the directory of any disk cataloged on the MASTER DIRECTORY DISK.

It will be able to print the directory in a long list (PRINT'DIR'REG) or as a multi-column list with many useful applications (PRINT'DIR'LABEL).

We can use many modules already on the disk unchanged: INTRO, INIT, PRINTER, PAGE, SCREEN, MENU, DUAL'DRIVE, and FILE'EXISTS. Now, you may see some benefits to modular programming. And in addition, PRINT'DIR'REG is adapted from READ'DIR, and PRINT'DIR'LABEL is taken from a program from COMAL TODAY #1 and COMMODORE MAGAZINE. Portability benefits continue.

We then only need to add a module to request what type of directory is needed (TYPE'OF'DIR), two that print the directories (PRINT'ALL and PRINT'IT), and a module to get the directory from the MASTER DIRECTORY DISK (GET'DIR).

Let's begin with the module to get the directory.

MODULE: GET'DIR

PURPOSE: Gets a disks directory from the MASTER DIRECTORY DISK.

```
NEW
AUTO 9000

9000 //
9010 PROC GET'DIR(NAME$,REF D$(),REF F'TYPE#(),REF F'BLOCKS#,REF COUNT)
```

Notice how entire arrays can be passed as parameters in reference.

```
9020  DIR'FILE:=8
9030  OPEN FILE DIR'FILE,NAME$,READ
9040  READ FILE DIR'FILE: DISK'ID$
9050  READ FILE DIR'FILE: BLOCKS'FREE
```

```
9060  READ FILE DIR'FILE: DISK'NAME$
9070  READ FILE DIR'FILE: COUNT
9080  READ FILE DIR'FILE: DATE$
9090  FOR X:=1 TO COUNT DO
```

The disk's summary told us how many files there were (COUNT). We now use this information to read that many file enteries.

```
9100   READ FILE DIR'FILE: D$(X),F'TYPE#(X),F'BLOCKS#(X)
9110  ENDFOR X
9120  CLOSE FILE DIR'FILE
9130 ENDPROC GET'DIR
```

Done. List it to disk: LIST "GET'DIR.L"

MODULE: TYPE'OF'DIR

PURPOSE: Asks what type of directory is needed.

```
NEW
AUTO 9000

9000 //
9010 PROC TYPE'OF'DIR(REF LABELS,REF NUM'COLS,REF NUM'ROWS) CLOSED
9020  DIM REPLY$ OF 1
9030  INPUT "MULTI-COLUMN DIRECTORY (Y/N): ": REPLY$
9040  IF REPLY$ IN "Yy" THEN
9050   LABELS:=TRUE
9060   INPUT "HOW MANY COLUMNS WIDE:2[LEFT]": NUM'COLS
```

Hit the CURSOR LEFT key in place of [LEFT]

```
9070   INPUT "HOW MANY LINES PER PAGE:8[LEFT]": NUM'ROWS
9080  ELSE
9090   LABELS:=FALSE; NUM'COLS:=1; NUM'ROWS:=0
9100  ENDIF
9110 ENDPROC TYPE'OF'DIR
```

Done. LIST it to disk: LIST "TYPE'OF'DIR.L"

```
                          --------------------
                          MODULE: PRINT'DIR'REG
                          --------------------

PURPOSE: Prints a regular single column directory.

NEW
AUTO 9000

9000 //
9010 PROC PRINT'DIR'REG(REF D$(),REF F'TYPE#(),REF F'BLOCKS#(),REF
       add this to end of above line:                              COUNT)
9020  PRINT "DISK:";DISK'NAME$;"ID:";DISK'ID$
9030  PRINT
9040  PRINT "NUM FILE NAME        TYP BLOCKS"
9050  PRINT "--- --------------- --- ------"
9060  FOR X:=1 TO COUNT DO
9070   PRINT USING "###": X;
9080   PRINT D$(X),TAB(22),TYPE$(F'TYPE#(X));
9090   PRINT USING "###": F'BLOCKS#(X)
9100  ENDFOR X
9110  PRINT COUNT;"FILES AND";BLOCKS'FREE;"BLOCKS FREE"
9120 ENDPROC PRINT'DIR'REG

Done. List it to disk: LIST "PRINT'DIR'REG.L"

                          ----------------------
                          MODULE: PRINT'DIR'LABEL
                          ----------------------

PURPOSE: Prints a multi-column directory.

NEW
AUTO 9000

9000 //
9010 PROC PRINT'DIR'LABEL(REF D$(),START,NUM'COLS,NUM'ROWS,REF
       add this to end of above line:                        FILE'COUNT)
9020  START:-1
9030  D$(0):=">>"+DISK'ID$+" "+DATE$+" "+DISK'ID$+"<<"
9040  FOR ROW:=0 TO NUM'ROWS-1 DO
9050   FOR COL:=0 TO NUM'COLS-1 DO
9060    THIS'ONE:=START+(COL*NUM'ROWS)+ROW
9070    IF THIS'ONE<=FILE'COUNT THEN PRINT TAB(1+(17*COL)),D$(THIS'ONE),
9080   ENDFOR COL
```

```
9090   PRINT
9100  ENDFOR ROW
9110  PRINT
```

The PRINT gives a blank line between 'pages'. You may wish to chage it to issue a form feed to your printer: PRINT CHR$(12),.

```
9120  START:+(NUM'COLS*NUM'ROWS)
9130  IF FILE'COUNT>=START THEN
9140   D$(START-1):=DISK'ID$+" CONTINUED "+DISK'ID$
```

This privides a continuation title on the next page. It overwrites the last file name printed. The next page goes back 1 file (see line 9020) to pick up this title.

```
9150   PRINT'DIR'LABEL(D$,START,NUM'COLS,NUM'ROWS,FILE'COUNT)
```

This is a recursive call. It will keep calling itself until all the file names are printed.

```
9160  ENDIF
9170 ENDPROC PRINT'DIR'LABEL
```

Done. List it to disk: LIST "PRINT'DIR'LABEL.L"

MODULE: PRINT'IT

PURPOSE: Check if directory file exists and then calls the print routine
MODULES REQUIRED: FILE'EXISTS, GET'DIR, PRINT'DIR'REG, PRINT'DIR'LABEL

```
NEW
AUTO 9000

9000 //
9010 PROC PRINT'IT(FILE'NAME$)
9020  IF FILE'EXISTS(FILE'NAME$) THEN
9030   GET'DIR(FILE'NAME$,D$,F'TYPE#,F'BLOCKS#,COUNT)
9040   IF LABELS THEN
9050    PRINT'DIR'LABEL(D$,1,NUM'COLS,NUM'ROWS,COUNT)
9060   ELSE
9070    PRINT'DIR'REG(D$,F'TYPE#,F'BLOCKS#,COUNT)
9080   ENDIF
9090  ENDIF
9100 ENDPROC PRINT'IT
```

Done. List it to disk: LIST "PRINT'IT.L"

```
                                ----------------
                                MODULE: PRINT'ALL
                                ----------------
```

PURPOSE: Cycles through all directories in the MASTER, printing them all
MODULE REQUIRED: PRINT'IT

```
NEW
AUTO 9000

9000 //
9010 PROC PRINT'ALL(MASTER$)
9020  ID'FILE:=9
9030  OPEN FILE ID'FILE,MASTER$+"DISK'IDS.DATA",READ
9040  WHILE NOT EOF(ID'FILE) DO
9050   READ FILE ID'FILE: ID$
9060   PRINT'IT(MASTER$+"DIRECTORY.."+ID$)
9070  ENDWHILE
9080  CLOSE FILE ID'FILE
9090 ENDPROC PRINT'ALL
```

Done. LIST it to disk: LIST "PRINT'ALL.L"

```
                              ==================
                              PROGRAM: PRINT'DIR
                              ==================
                              FINAL CONSTRUCTION
```

OLD MODULES: INTRO, INIT, PAGE, PRINTER, SCREEN, MENU, DUAL'DRIVE,
FILE'EXISTS
NEW MODULES: PRINT'IT, PRINT'ALL, PRINT'DIR'LABEL, PRINT'DIR'REG,
GET'DIR, TYPE'OF'DIR

All the modules needed are now on the disk. Now we just enter the main program, then MERGE in the modules.

```
NEW
AUTO

0010 //DELETE "0:PRINT'DIR1"
0020 //(C)1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:PRINT'DIR3"
0040 INTRO("PRINT DISK DIRECTORY")
```

```
0050 MAX'FILES:=144
0060 DIM D$(0:MAX'FILES) OF 16, F'TYPE#(1:MAX'FILES), F'BLOCKS#(1:
       add this to above line:                              MAX'FILES)
```

Note that the array D$, for the file names on the disk starts at 0 instead of 1. This allows for the title record added in the PRINT'DIR'LABEL module.

```
0070 DIM DISK'ID$ OF 2, MASTER$ OF 2
0080 DIM DISK'NAME$ OF 16, DATE$ OF 6, FIND'ID$ OF 2
0090 DIM ID$ OF 2
0100 DIM TYPE$(129:132) OF 3
0110 TYPE$(129):="SEQ"; TYPE$(130):="PRG"
0120 TYPE$(131):="USR"; TYPE$(132):="REL"
0130 INIT(MASTER$)
0140 PRINT "WHICH DISK ID TO PRINT (OR * FOR ALL):"
0150 INPUT FIND'ID$
0160 TYPE'OF'DIR(LABELS,NUM'COLS,NUM'ROWS)
0170 IF PRINTER THEN SELECT OUTPUT "LP:"
0180 IF FIND'ID$="*" THEN
0190  PRINT'ALL(MASTER$)
0200 ELSE
0210  PRINT'IT(MASTER$+"DIRECTORY.."+FIND'ID$)
0220 ENDIF
0230 SCREEN
0240 MENU
```

Now just MERGE in the modules already on disk:

```
ENTER "PRINT'IT.L"
RENUM
ENTER "INTRO.L"
RENUM
ENTER "INIT.L"
RENUM
ENTER "PRINTER.L"
RENUM
ENTER "PAGE.L"
RENUM
ENTER "SCREEN.L"
RENUM
ENTER "MENU.L"
RENUM
ENTER "DUAL'DRIVE.L"
RENUM
ENTER "PRINT'ALL.L"
```

```
RENUM
ENTER "FILE'EXISTS.L"
RENUM
ENTER "PRINT'DIR'REG.L"
RENUM
ENTER "PRINT'DIR'LABEL.L"
RENUM
ENTER "GET'DIR.L"
RENUM
ENTER "TYPE'OF'DIR.L"
RENUM
```

That's it. Program construction is complete. Now just save it to disk.

```
==================
PROGRAM: PRINT'IDS
==================
 PRELIMINARY WORK
```

This program will print all the disk IDs currently in use. It will print them as a chart (PRINT'ID'CHART) or in a list (PRINT'ID'LIST).

We only need to add a module to find out what whether a LIST or a CHART is desired (TYPE'OF'LIST), a module to set up a line for the CHART (FIX'LINE) and a module to skip any IDs not in our CHART listing (SKIP'IDS).

Let's begin with the module to find what kind of output is wanted:

```
--------------------
MODULE: TYPE'OF'LIST
--------------------
```

PURPOSE: Find out whether a LIST or a CHART is wanted.
MODULES REQUIRED: PRINT'ID'CHART, PRINT'ID'LIST

```
NEW
AUTO 9000

9000 //
9010 PROC TYPE'OF'LIST
9020  INPUT "LIST OR CHART (L/C):": REPLY$
9030  IF REPLY$="C" OR REPLY$="c" THEN
9040   INPUT "40 COL LIMIT?": REPLY$
9050   IF REPLY$ IN "Yy" THEN
9060    PRINT'ID'CHART(33,31)
```

We pass two parameters to the PRINT'ID'CHART procedure. The first parameter is the ordinal value of the starting character for the row. The other parameter specifies how many characters more to include in the row (twice as many when 40 columns is not a limit).

```
9070    PRINT'ID'CHART(65,31)
9080   ELSE
9090    PRINT'ID'CHART(33,62)
9100   ENDIF
9110  ELSE
9120   PRINT'ID'LIST
9130  ENDIF
9140 ENDPROC TYPE'OF'LIST
```

Done. LIST it to disk: LIST "TYPE'OF'LIST.L"

```
                        ---------------------
                        MODULE: PRINT'ID'LIST
                        ---------------------
```

PURPOSE: Print the disk IDs in a long list.

```
NEW
AUTO 9000

9000 //
9010 PROC PRINT'ID'LIST
9020  FOR X:=1 TO COUNT DO PRINT ID$(X)
9030 ENDPROC PRINT'ID'LIST
```

Done. LIST it to disk: LIST "PRINT'ID'LIST.L"

```
                        ----------------------
                        MODULE: PRINT'ID'CHART
                        ----------------------
```

PURPOSE: Print disk IDs in a chart, allowing for either 40 or 80 columns
MODULES REQUIRED: SKIP'IDS, FIX'LINE

```
NEW
AUTO 9000

9000 //
9010 PROC PRINT'ID'CHART(COL'START,ADD'COL)
```

For parameters, we use the ordinal value of the starting ID (the column in the chart) and the number of additional columns (characters) to print across.

```
9020  CURRENT'ID:=1
9030  IF ORD(ID$(CURRENT'ID))<33 THEN SKIP'IDS
9040  HEADING$:=" "
9050  FOR X=COL'START TO COL'START+ADD'COL DO HEADING$:=HEADING$+CHR$(X)
```

HEADING$ is used as a divider line between rows. It is an informative line rather than just dashes -----.

```
9060  FOR ROW:=33 TO 95 DO
9070   IF NOT ((ROW-33) MOD 5) THEN PRINT HEADING$
```

This prints the dividing HEADING line after every 5 lines in the chart. Change the 5 to another number to make the HEADING line more or less frequent.

```
9080   LINE$:=CHR$(ROW) // START OF NEXT CHART LINE IS ITS ID CHARACTER
9090   FOR COLS:=2 TO ADD'COL+2 DO LINE$(COLS):=FILLER$
```

Builds the next chart line with the default of no IDs used in that line. FILLER$ can vary depending on whether the output is on the screen or the printer, or you may use other criteria as a modification.

```
9100   IF CURRENT'ID<=COUNT THEN
9110    IF ROW=ORD(ID$(CURRENT'ID)) THEN FIX'LINE
```

If an ID is used in the line, FIX'LINE is called to plot it into the line.

```
9120   ENDIF
9130   PRINT LINE$
9140  ENDFOR ROW
9150 ENDPROC PRINT'ID'CHART
```

Done. LIST it to disk: LIST "PRINT'ID'CHART.L"

MODULE: FIX'LINE

PURPOSE: Fix next chart line to include a # at the spot of an ID in use

```
NEW
```

```
AUTO 9000

9000 //
9010 PROC FIX'LINE
9020  ORD2:=ORD(ID$(CURRENT'ID)(2))
```

This shows how to take a substring of an element in a string array. We are looking only at the second character of the current ID$.

```
9030  IF ORD2>=COL'START AND ORD2<=COL'START+ADD'COL THEN
9040   LINE$(2+ORD2-COL'START):="#"
9050  ENDIF
9060  CURRENT'ID:+1
9070  IF CURRENT'ID<=COUNT THEN
9080   IF ORD(ID$(CURRENT'ID))=ROW THEN FIX'LINE
```

This line keeps calling FIX'LINE as long as another ID starts with the same character (recursive).

```
9090  ENDIF
9100 ENPROC FIX'LINE
```

Done. LIST it to disk: LIST "FIX'LINE.L"

MODULE: SKIP'IDS

PURPOSE: Skips over any IDs not included in the range of our ID CHART.

```
NEW
AUTO 9000

9000 //
9010 PROC SKIP'IDS
9020  CURRENT'ID:+1
9030  IF CURRENT'ID<=COUNT THEN
9040   IF ORD(ID$(CURRENT'ID))<33 THEN SKIP'IDS
```

This keeps calling itself as long as there may be more IDs to skip (recursive).

```
9050  ENDIF
9060 ENDPROC SKIP'IDS
```

Done. LIST it to disk: LIST "SKIP'IDS.L"

```
                                ==================
                                PROGRAM: PRINT'IDS
                                ==================
                                FINAL CONSTRUCTION

OLD MODULES: READ'IDS, SORT'IDS, WRITE'SORTED, INTRO, PAGE, INIT,
              DUAL'DRIVE, PRINTER, SCREEN, MENU, three QUICKSORT modules
NEW MODULES: PRINT'ID'CHART, PRINT'ID'LIST, SKIP'IDS, FIX'LINE

All the modules needed by the program are now on the disk.  We  now  can
write the main program and then MERGE in the modules from disk.

NEW
AUTO

0010 //DELETE "0:PRINT'IDS1"
0020 //(C)1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:PRINT'IDS3"
0040 INTRO("PRINT LIST OR CHART OF DISK IDS IN USE")
0050 MAX'NUM'DISKS:=140
0060 DIM ID$(1:MAX'NUM'DISKS) OF 2, LAST'ID$ OF 2, LINE$ OF 64
0070 DIM HEADING$ OF 64, FILLER$ OF 1, REPLY$ OF 1, MASTER$ OF 2
0080 INIT(MASTER$)
0090 READ'IDS
0100 SORT'IDS
0110 WRITE'SORTED
0120 IF PRINTER THEN
0130  SELECT OUTPUT "LP:"
0140  FILLER$:=" "

For faster printing spaces are used. This also saves the  printwheel  on
daisy wheel printers.

0150 ELSE
0160  FILLER$:=":"
0170 ENDIF
0180 TYPE'OF'LIST
0190 SCREEN
0200 MENU

Now just MERGE in the modules:

ENTER "READ'IDS.L"
RENUM
ENTER "SORT'IDS.L"
```

```
RENUM
ENTER "WRITE'SORTED.L"
RENUM
ENTER "PRINT'ID'CHART.L"
RENUM
ENTER "PRINT'ID'LIST.L"
RENUM
ENTER "SKIP'IDS.L"
RENUM
ENTER "FIX'LINE.L"
RENUM
ENTER "INTRO.L"
RENUM
ENTER "PAGE.L"
RENUM
ENTER "INIT.L"
RENUM
ENTER "DUAL'DRIVE.L"
RENUM
ENTER "PRINTER.L"
RENUM
ENTER "SCREEN.L"
RENUM
ENTER "MENU.L"
RENUM
ENTER "QUICKSORT.L"
RENUM
ENTER "TYPE'OF'LIST.L"
RENUM
```

That's it. Program construction is complete. Save the program. You know how.

PROGRAM: FIND'FILE

PRELIMINARY WORK

This program will search through the MASTER FILE for the file name you specify (or you can use a wild card search utilizing the * character as with Commodore's disk operating system). The file names matching your search name will be listed along with the ID of the disk that they are stored on.

We will be using some modules from previous programs: MENU, SCREEN, PRINTER, DUAL'DRIVE, PAGE, INIT, and INTRO (these make up over half the

program). We only need to write the modules that request the name to search for (FIND'WHAT), a module to do the searching (SEARCH'PRINT), one to print the files that match (PRINT'IT2), and a module to get a Y or N answer. We'll be done with this program one, two, three.

```
-----------------
MODULE: FIND'WHAT
-----------------
```

PURPOSE: Request a name to search for. Adjusts for a * wild card search
MODULE REQUIRED: PAGE

```
NEW
AUTO 9000

9000 //
9010 PROC FIND'WHAT
9020  PAGE
9030  PRINT "ENTER THE NAME OF THE FILE TO FIND"
9040  PRINT "USE A * TO MEAN IGNORE REST OF THE NAME"
9050  PRINT "ENTER JUST A * TO LIST ALL FILES"
9060  PRINT
9070  INPUT "WHAT NAME TO SEARCH FOR:": FIND'NAME$
9080  IF "*" IN FIND'NAME$ THEN
9090   END'POS:=("*" IN FIND'NAME$)-1
```

If a * was included in the name, we set the length (END'POS) to just before the * occurred, since we do not need to match characters from that position on.

```
9100  ELSE
9110   END'POS:=16
9120   FIND'NAME$:=FIND'NAME$+"                "
```

File names are stored as 16 character long names. If a name is less than 16 characters, it is "padded" with spaces. The above line adds 16 spaces to the name to make sure it is long enough in every case.

```
9130  ENDIF
9140 ENDPROC FIND'WHAT
```

Done. List it to disk: LIST "FIND'WHAT.L"

```
                    --------------------
                    MODULE: SEARCH'PRINT
                    --------------------

PURPOSE: Searches through the MASTER FILE, printing any names that match
MODULE REQUIRED: PRINT'IT2

NEW
AUTO 9000

9000 //
9010 PROC SEARCH'PRINT
9020  ALL'FILE:=5
9030  OPEN FILE ALL'FILE,MASTER$+"ALLFILES.DATA",READ
9040  WHILE NOT EOF(ALL'FILE) DO
9050   READ FILE ALL'FILE: FILE'NAME$
9060   IF END'POS=0 THEN
9070    PRINT'IT2
```

If the user specified a search name of only "*" then all files are to be printed. In that case the variable END'POS was set to equal 0 - no characters need to match. So in that case, we print every name read.

```
9080   ELIF FIND'NAME$(1:END'POS)=FILE'NAME$(1:END'POS) THEN
9090    PRINT'IT2
9100   ENDIF
9110  ENDWHILE
9120  CLOSE FILE ALL'FILE
9130 ENDPROC SEARCH'PRINT

Done. List it to disk: LIST "SEARCH'PRINT.L"

                    -----------------
                    MODULE: PRINT'IT2
                    -----------------
```

PURPOSE: Prints the file name just matched with the ID of its disk

We call this module PRINT'IT2, since we already have a PRINT'IT module in a previous program.

```
NEW
AUTO 9000

9000 //
```

```
9010 PROC PRINT'IT2
9020  PRINT FILE'NAME$(1:16);"DISK:";FILE'NAME$(17:18)
```

Remember? The disk id is stored in the file name as characters 17 and 18.

```
9030 ENDPROC PRINT'IT2
```

Done. List it to disk: LIST "PRINT'IT2.L"

MODULE: VERIFIED

PURPOSE: Prints the prompt plus (Y/N), and only accepts a Y or N answer

```
NEW
AUTO 9000

9000 //
9010 PROC VERIFIED(PROMPT$) CLOSED
9020  PRINT PROMPT$+" (Y/N):",
9030  REPEAT
9040   CASE KEY$ OF
9050   WHEN "Y","y"
9060    FOR X:=1 TO LEN(PROMPT$)+7 DO PRINT CHR$(20),
```

This line erases the prompt from the screen. This is not useful now, but in your future programs you may find it is just what you need.

```
9070    RETURN TRUE
9080   WHEN "N","n"
9090    FOR X:=1 TO LEN(PROMPT$)+7 DO PRINT CHR$(20),
9100    RETURN FALSE
9110   OTHERWISE
9120    NULL
9130   ENDCASE
9140  UNTIL TRUE=FALSE // FOREVER
9150 ENDFUNC VERIFIED
```

Done. List it to disk: LIST "VERIFIED.L"

==========
PROGRAM: FIND'FILE
==========
FINAL CONSTRUCTION

OLD MODULES: INTRO, INIT, PAGE, DUAL'DRIVE, PRINTER, SCREEN, MENU
NEW MODULES: VERIFIED, FIND'WHAT, SEARCH'PRINT, PRINT'IT2

All the modules needed by the program are now on the disk. We just write the main program, and then MERGE in the modules.

```
NEW
AUTO

0010 //DELETE "0:FIND'FILE1"
0020 //(C)1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:FIND'FILE3"
0040 INTRO("FIND WHAT DISKS CONTAIN A SPECIFIC FILE")
0050 DIM FILE'NAME$ OF 19, REPLY$ OF 1, MASTER$ OF 2, FIND'NAME$ OF 16
0060 INIT(MASTER$)
0070 REPEAT
0080  FIND'WHAT
0090  IF PRINTER THEN SELECT OUTPUT "LP:"
0100  SEARCH'PRINT
0110  SCREEN
0120 UNTIL NOT VERIFIED("MORE TO FIND")
0130 MENU
```

That's the main program. Now just MERGE in the modules:

```
ENTER "FIND'WHAT.L"
RENUM
ENTER "SEARCH'PRINT.L"
RENUM
ENTER "PRINT'IT2.L"
RENUM
ENTER "INTRO.L"
RENUM
ENTER "INIT.L"
RENUM
ENTER "PAGE.L"
RENUM
ENTER "DUAL'DRIVE.L"
RENUM
ENTER "PRINTER.L"
```

```
RENUM
ENTER "SCREEN.L"
RENUM
ENTER "MENU.L"
RENUM
ENTER "VERIFIED.L"
```

That's it. Program construction is complete. Save the program. You know how.

PROGRAM: DELETE'DIR

PRELIMINARY WORK

This program will delete disks from the MASTER. Just specify the disk's ID and the program will first report the summary for the disk, and verify that this is the correct disk to delete. This program is useful when you remove disks from your library (involuntary sometimes - a ruined disk).

We only will have to write one new module for the program (DELETE'IT). All the rest of the modules will be borrowed from previous programs. You now should begin to understand the real advantage of modular programming.

MODULE: DELETE'IT

PURPOSE: This module deletes a disk from the MASTER after verification
MODULE REQUIRED: FILE'EXISTS

```
NEW
AUTO 9000

9000 //
9010 PROC DELETE'IT(MASTER$,FILE'NAME$,REF UPDATED'FLAG, DELETE'ID$)
       add this to end of above line:                          CLOSED
9020  DIM DISK'ID$ OF 2, DISK'NAME$ OF 16, DATE$ OF 6, REPLY$ OF 1
9030  DIM TEMP'ID$ OF 2
9040  IF FILE'EXISTS(MASTER$+FILE'NAME$) THEN
9050   DIR'FILE:=8
9060   OPEN FILE DIR'FILE,MASTER$+FILE'NAME$,READ
9070   READ FILE DIR'FILE: DISK'ID$,BLOCKS,DISK'NAME$,FILE'COUNT,DATE$
9080   CLOSE FILE DIR'FILE
```

```
9090   PRINT DISK'ID$;DISK'NAME$;"CATALOGED ON";DATE$
9100   PRINT FILE'COUNT;"FILES";"WITH";BLOCKS;"BLOCKS FREE"
9110   INPUT "DELETE THIS DIRECTORY FROM MASTER?": REPLY$
9120   IF REPLY$="Y" OR REPLY$="y" THEN
9130    DELETE MASTER$+FILE'NAME$
```

Here we delete the directory file. Next we remove the ID from the DISK ID file.

```
9140    ID'FILE:=9; NEW'FILE:=10
9150    OPEN FILE ID'FILE,MASTER$+"DISK'IDS.DATA",READ
9160    DELETE MASTER$+"NEW'IDS.DATA"
9170    OPEN FILE NEW'FILE,MASTER$+"NEW'IDS.DATA",WRITE
9180    WHILE NOT EOF(ID'FILE) DO
9190     READ FILE ID'FILE: TEMP'ID$
9200     IF TEMP'ID$<>DELETE'ID$ THEN WRITE FILE NEW'FILE: TEMP'ID$
9210    ENDWHILE
9220    CLOSE FILE ID'FILE
9230    CLOSE FILE NEW'FILE
```

The next three lines delete the backup ID file, then rename the current file to the old file, and finally rename the new file to the current file.

```
9240    DELETE MASTER$+"OLD'IDS.DATA"
9250    PASS "R"+MASTER$+"OLD'IDS.DATA=DISK'IDS.DATA"
9260    PASS "R"+MASTER$+"DISK'IDS.DATA=NEW'IDS.DATA"
9270    UPDATED'FLAG:=TRUE
```

You remember the updated flag of course.

```
9280   ENDIF
9290  ELSE
9300   PRINT FILE'NAME$;"DOES NOT EXIST"
9310  ENDIF
9320 ENDPROC DELETE'IT
```

Done. List it to disk: LIST "DELETE'IT.L"

```
                                ==================
                                PROGRAM: DELETE'DIR
                                ==================
                                FINAL CONSTRUCTION

OLD MODULES: SET'UPDATED, INTRO, PAGE, INIT, DUAL'DRIVE, FILE'EXISTS,
              MENU2
NEW MODULE:  DELETE'IT
```

All the modules needed by the program are now on the disk. Next we write the main program and MERGE in the modules.

```
NEW
AUTO

0010 //DELETE "0:DELETE'DIR1"
0020 //(C)1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:DELETE'DIR3"
0040 INTRO("DELETE A DISK ENTRY FROM THE MASTER DISK")
0050 DIM MASTER$ OF 2, ID$ OF 2
0060 UPDATED'FLAG=FALSE
0070 INIT(MASTER$)
0080 REPEAT
0090  INPUT "WHICH DISK ID TO DELETE (*=QUIT): ": ID$
0100  IF ID$<>"*" THEN DELETE'IT(MASTER$,"DIRECTORY.."+ID$,
        add this to line above:                    UPDATED'FLAG,ID$)
0110 UNTIL ID$="*"
0120 MENU2(UPDATED'FLAG)
```

Now just MERGE in the modules:

```
ENTER "DELETE'IT.L"
RENUM
ENTER "SET'UPDATED.L"
RENUM
ENTER "INTRO.L"
RENUM
ENTER "PAGE.L"
RENUM
ENTER "INIT.L"
RENUM
ENTER "DUAL'DRIVE.L"
RENUM
ENTER "FILE'EXISTS.L"
RENUM
```

```
ENTER "MENU2.L"
RENUM
```

That's it. Program construction is complete. Save the program. You know how.

==========
PROGRAM: COMPARE'DIR
==========
PRELIMINARY WORK

This program will compare the directories of any two disks in the MASTER. This is useful to find the one program that is different, or the one that is the same, and all the many cases in between. A few years ago when I needed this program I didn't have it. Now I do, and you do too.

Once again over half of the program is already written, using our old modules. We will tailor the GET'DIR module to this program, creating GET'DIR2. The new modules to be written are those to get the two disk ID's (GET'ID), get a directory and sort it (DIRECTORY and SORT'DIR), compare two directories (COMPARE), and the modules to do the printing of the two directories (PRINT'D0, PRINT'D1, and PRINT'BOTH).

First let's modify the GET'DIR module we wrote as part of the PRINT'DIR program:

MODULE: GET'DIR2

PURPOSE: Gets a disks directory from the MASTER DIRECTORY DISK.

The quickest way to modify the GET'DIR module already written is to retrieve it from disk, and then edit it:

```
NEW
ENTER "GET'DIR.L"

EDIT

9000 //
9010 PROC GET'DIR(NAME$,REF D$(),REF F'TYPE#(),REF F'BLOCKS#,REF COUNT)
9020  DIR'FILE:=8
9030  OPEN FILE DIR'FILE,NAME$,READ
9040  READ FILE DIR'FILE: DISK'ID$
9050  READ FILE DIR'FILE: BLOCKS'FREE
```

```
9060  READ FILE DIR'FILE: DISK'NAME$
9070  READ FILE DIR'FILE: COUNT
9080  READ FILE DIR'FILE: DATE$
9090  FOR X:=1 TO COUNT DO
9100   READ FILE DIR'FILE: D$(X),F'TYPE#(X),F'BLOCKS#(X)
9110  ENDFOR X
9120  CLOSE FILE DIR'FILE
9130 ENDPROC GET'DIR
```

Of course we must change the proc name in the header and ENDPROC. Cursor up to line 9010 to fix the header. Then cursor back down to the ENDPROC and add a 2 to the end of the name there too.

In the COMPARE'DIR program, we don't need to know the file type and blocks for every file on the disk. So we will skip them. But since they still are part of the disk files we will be reading, the way we will do this is to "dummy them out". This is strange terminology IBM mainframe (the big million dollar computers) programmers will recognize from their JCL coding. This is how it works: We simply change the references to the arrays and variables we wish to skip to be the variables DUMMY$ (to skip string variables) and DUMMY2# (to skip integer variables or arrays). We can't use DUMMY# because a variable name can only be used once, and DUMMY is already used for a string variable. Now start by changing the two integer array names in the header to DUMMY2#:

```
EDIT 9010
```

```
9010 PROC GET'DIR2(NAME$,REF D$(),REF F'TYPE#(),REF F'BLOCKS#,REF COUNT)
```

Now edit to look like this:

```
9010 PROC GET'DIR(NAME$,REF D$(),DUMMY2#,DUMMY2#,REF COUNT)
```

Next we must make similar changes to line 9100:

```
EDIT 9100:
```

```
9100   READ FILE DIR'FILE: D$(X),F'TYPE#(X),F'BLOCKS#(X)
```

Change it to:

```
9100   READ FILE DIR'FILE: D$(X),DUMMY2#,DUMMY2#
```

Finally, we also can "dummy out" the variables DISK'ID$, DISK'NAME$, and DATE$ since we do not need them in this program either. Edit lines 9040, 9060, and 9080 to look like this:

```
9040  READ FILE DIR'FILE: DUMMY$

9060  READ FILE DIR'FILE: DUMMY$

9080  READ FILE DIR'FILE: DUMMY$
```

Now, the modified module is done. List it to disk: LIST "GET'DIR2.L"

```
                              --------------
                              MODULE: GET'ID
                              --------------
```

```
PURPOSE: Gets an ID of a disk to use for the COMPARE
MODULE REQUIRED: FILE'EXISTS

NEW
AUTO 9000

9000 //
9010 PROC GET'ID(REF ID$) CLOSED
9020  REPEAT
9030   INPUT "DISK ID:": ID$;
9040   PRINT "-->";
9050   IF FILE'EXISTS("DIRECTORY.."+ID$) THEN
9060    DONE:=TRUE
9070    PRINT "OK"
9080   ELSE
9090    DONE:=FALSE
9100    PRINT "DIRECTORY NOT FOUND"
9110   ENDIF
9120  UNTIL DONE
9130 ENDPROC GET'ID
```

Done. List it to disk: LIST "GET'ID.L"

```
                              -----------------
                              MODULE: DIRECTORY
                              -----------------
```

```
PURPOSE: Gets the directory from the MASTER and sorts it
MODULES REQUIRED: GET'DIR2, SORT'DIR

NEW
AUTO 9000
```

```
9000 //
9010 PROC DIRECTORY(REF D$(),ID$)
9020  FILE'COUNT:=0
9030  FOR X:=1 TO MAX'FILES DO D$(X):=""
```

The above line initializes the array of file names to be all blank.

```
9040  GET'DIR2(MASTER$+"DIRECTORY.."+ID$,D$,DUMMY2#,DUMMY2#,FILE'COUNT)
9050  SORT'DIR(D$,FILE'COUNT)
9060 ENDPROC DIRECTORY
```

Done. List it to disk: LIST "DIRECTORY.L"

MODULE: SORT'DIR

PURPOSE: Sort the directory
MODULES REQUIRED: three QUICKSORT modules

```
NEW
AUTO 9000

9000 //
9010 PROC SORT'DIR(REF D$(),COUNT) CLOSED
9020  PRINT "SORTING DIRECTORY..."
9030  QUICKSORT(D$,1,COUNT,16)
9040 ENDPROC SORT'DIR
```

Done. List it to disk: LIST "SORT'DIR.L"

MODULE: COMPARE

PURPOSE: Compares the two directories
MODULES REQUIRED: PRINT'D0, PRINT'D1, PRINT'BOTH

```
NEW
AUTO 9000

9000 //
9010 PROC COMPARE(REF D0$(),REF D1$())
```

Always pass arrays as parameters in REFerence. It saves memory. And besides, version 0.14 COMAL requires it.

```
9020  PRINT
9030  PRINT "DISK";ID0$,TAB(18),"BOTH",TAB(33),"DISK";ID1$
9040  PRINT "-------",TAB(18),"----",TAB(33),"-------"
9050  D0'DONE:=FALSE; D1'DONE:=FALSE; D0'COUNT:=1; D1'COUNT:=1
9060  REPEAT
9070   IF D0'DONE THEN
9080    PRINT'D1
```

If directory 0 has been exhausted, we merely keep printing directory 1 until it is done. The same goes for directory 1 below:

```
9090   ELIF D1'DONE THEN
9100    PRINT'D0
9110   ELIF D0$(D0'COUNT)=D1$(D1'COUNT) THEN
9120    PRINT'BOTH
```

If both names are the same, print the name in the both column.

```
9130   ELIF D1$(D1'COUNT)<D0$(D0'COUNT) THEN
9140    PRINT'D1
```

If the file in directory 1 is compares to be less than the directory 0 file then PRINT'D1. If the reverse is true, as below, then PRINT'D0:

```
9150   ELIF D0$(D0'COUNT)<D1$(D1'COUNT) THEN
9160    PRINT'D0
```

If we get to this point something is wrong:

```
9170   ELSE
9180    PRINT "ERROR IN COMPARE"
9190    D0'DONE:=TRUE; D1'DONE:=TRUE
```

Do you know why we just don't put an END statement here? Why set the end o directory flags to TRUE instead?

The output may be going to the printer. We will want to reset it back to the screen. And we are inside a SERIES of programs. Ending abruptly will stop the whole series. A graceful ending still allows the MENU to be called again.

```
9200   ENDIF
9210  UNTIL D0'DONE AND D1'DONE
9220 ENDPROC COMPARE
```

Done. List it to disk: LIST "COMPARE.L"

MODULE: PRINT'D0

PURPOSE: Print a line with the file name in the D0 column and update D0

```
NEW
AUTO 9000

9000 //
9010 PROC PRINT'D0
9020  PRINT D0$(D0'COUNT),TAB(18),"...",TAB(37),"..."
9030  D0'COUNT:+1
9040  IF D0$(D0'COUNT)="" THEN D0'DONE:=TRUE
9050 ENDPROC PRINT'D0
```

Done. List it to disk: LIST "PRINT'D0.L"

MODULE: PRINT'D1

PURPOSE: Print a line with the file name in the D1 column and update D1

You may have thought that this proc could be easily modified from PRINT'D0. Not really. We are being fancy and right justifying the names along the right side. This requires some extra calculations. The line printed is different, of course. So let's just write it from scratch:

```
NEW
AUTO 9000

9000 //
9010 PROC PRINT'D1
9020  TEMP$:=D1$(D1'COUNT); LENGTH:=LEN(TEMP$)
9030  WHILE TEMP$(LENGTH)=CHR$(32) DO LENGTH:-1
9040  PRINT "...",TAB(18),"...",TAB(39-LENGTH+1),TEMP$(1:LENGTH)
9050  D1'COUNT:+1
9060  IF D1$(D1'COUNT)="" THEN D1'DONE:=TRUE
9070 ENDPROC PRINT'D1
```

Done. List it to disk: LIST "PRINT'D1.L"

```
-----------------
MODULE: PRINT'BOTH
-----------------
```

PURPOSE: Print a line with the name in the BOTH column. Update D0 and D1

Now you can modify the previous PRINT'D1 module to fit here. Being fancy, of course we want to center the file name in the middle BOTH column. So we will be able to reuse the first two lines. The print line will change, and we must update both counts, and check both directories for their end.

```
NEW
ENTER "PRINT'D1.L"

EDIT

9000 //
9010 PROC PRINT'D1
9020 TEMP$:=D1$(D1'COUNT); LENGTH:=LEN(TEMP$)
9030 WHILE TEMP$(LENGTH)=CHR$(32) DO LENGTH:-1
9040 PRINT "...",TAB(18),"...",TAB(39-LENGTH+1),TEMP$(1:LENGTH)
9050 D1'COUNT:+1
9060 IF D1$(D1'COUNT)="" THEN D1'DONE:=TRUE
9070 ENDPROC PRINT'D1
```

Now change the name in the header and ENDPROC:

```
9010 PROC PRINT'BOTH
9070 ENDPROC PRINT'BOTH
```

Then add to line 9050 to increment the count for both:

```
9050 D0'COUNT:+1; D1'COUNT:+1
```

Add a line to check if directory 0 is done:

```
9055 IF D0$(D0'COUNT)="" THEN D0'DONE:=TRUE
```

Finally fix the print line:

```
9040 PRINT "...",TAB(19-(LENGTH DIV 2)),TEMP$(1:LENGTH),TAB(37),"..."
```

Done. Now RENUM and LIST it to disk:

```
RENUM 9000
LIST "PRINT'BOTH.L"

                              ====================
                              PROGRAM: COMPARE'DIR
                              ====================
                               FINAL CONSTRUCTION

OLD MODULES: INTRO, INIT, PRINTER, SCREEN, MENU, PAGE, FILE'EXISTS,
             DUAL'DRIVE, three QUICKSORT modules
MODIFIED MODULES: GET'DIR2 (FROM GET'DIR)
NEW MODULES: DIRECTORY, SORT'DIR, COMPARE, PRINT'D0, PRINT'D1,
             PRINT'BOTH, GET'ID
```

All the modules needed by the program are on disk. Now we write the main program and then MERGE in the modules:

```
NEW
      AUTO

0010 //DELETE "0:COMPARE'DIR1"
0020 //(C)1984 COMAL USERS GROUP, U.S.A., LIMITED
0030 //SAVE   "0:COMPARE'DIR3"
0040 INTRO("COMPARE DIRECTORIES OF ANY TWO DISKS")
0050 MAX'FILES:=125
```

Due to all our fancy ideas, we only have enough memory to hold 125 files. Remember, that is two directories of 125 files each - alot of storage required. Now you know why we "dummied out" the integer arrays.

```
0060 DIM DISK0$(1:MAX'FILES) OF 16, DISK1$(1:MAX'FILES) OF 16
0070 DIM ID0$ OF 2, ID1$ OF 2
0080 DIM DUMMY$ OF 1, MASTER$ OF 2
0090 DIM TEMP$ OF 16
0100 INIT(MASTER$)
0110 GET'ID(ID0$)
0120 PRINT
0130 GET'ID(ID1$)
0140 DIRECTORY(DISK0$,ID0$)
0150 DIRECTORY(DISK1$,ID1$)
0160 IF PRINTER THEN SELECT OUTPUT "LP:"
0170 COMPARE(DISK0$,DISK1$)
0180 SCREEN
0190 MENU
```

Now just MERGE in the modules:

```
ENTER "DIRECTORY.L"
RENUM
ENTER "SORT'DIR.L"
RENUM
ENTER "COMPARE.L"
RENUM
ENTER "PRINT'D0.L"
RENUM
ENTER "PRINT'D1.L"
RENUM
ENTER "PRINT'BOTH.L"
RENUM
ENTER "GET'ID.L"
RENUM
ENTER "INTRO.L"
RENUM
ENTER "INIT.L"
RENUM
ENTER "PRINTER.L"
RENUM
ENTER "SCREEN.L"
RENUM
ENTER "MENU.L"
RENUM
ENTER "PAGE.L"
RENUM
ENTER "FILE'EXISTS.L"
RENUM
ENTER "DUAL'DRIVE.L"
RENUM
ENTER "QUICKSORT.L"
RENUM
ENTER "GET'DIR2.L"
RENUM
```

That's it. Program construction is complete. Save the program. You know how.

APPENDIX A

HOW TO BACKUP A DISK

With a dual drive (ie, 4040):

```
        Backup drive 0 to drive 1:
COMAL:  PASS "d1=0"
BASIC:  OPEN 15,8,15,"d1=0"
        CLOSE 15
```

With a single drive (ie, 1541):

Use the 1541 BACKUP program on Commodore's BONUS PAK disk. If you don't have this program, a similar program, 1541 BACKUP(FREE), is included at the end of COMAL Users Group, U.S.A., Limited disks. Since a disk backup program needs as much free memory as possible, the program is written in BASIC. Do NOT attempt to LOAD it into COMAL.

HOW TO USE THE 1541 BACKUP(FREE) PROGRAM:

1) Remove any disks from the disk drive. Turn off computer.
2) Turn on computer and disk drive.

>>> Use unshifted letters when typing <<<

3) Type in this line followed by RETURN key:
   load "1541backup(free)",8
4) Type in this line followed by RETURN key:
   run
5) The screen clears and you are asked:
   disk name ?
6) Before replying, remove the original disk from the drive and insert a blank disk. The program will erase and format the disk for use as your backup disk.
7) Type in a name for the disk (up to 16 characters)
8) Next you are asked:
   unique disk id ?

Every disk has its own ID. This ID can be any two characters. No two disks should ever have the same disk ID. It is hard to emphasize this enough. The CAPTAIN COMAL DISK ORGANIZATION SYSTEM will help you keep track of your disk ID's. Type in two characters that you are SURE are not used elsewhere (when in doubt try something odd like [[).

9) The following instruction no appears:

insert source disk, press SPACE

10) REMOVE the new disk from the drive and label it with the disk name and ID. Insert the original disk, also called the SOURCE DISK. After doing this hit the space bar.
11) Information about the disk will scroll by. It's not important to read it. It ends by estimating about how long in minutes and seconds that the copy may take. Then the program counts and reads blocks from the SOURCE DISK. The count is flashed on the bottom screen line:
reading block #---

12) After 124 blocks are read you see this:
insert destination disk, press SPACE

13) Take out the original SOURCE DISK. Put the new DESTINATION DISK into the drive. After you do this hit the space bar.
14) As blocks are written the count is shown:
writing buffer #---

15) After all blocks are written you'll see:
insert source disk, press SPACE

16) Take out the new DESTINATION DISK. Put the original SOURCE DISK back into the drive. After you do this hit the space bar.
17) Now the familiar message appears again, flashing through the next set of blocks, 1 through 124 (yes, the numbers are 1 - 124 again, but this is a different set of 124 blocks):
reading block #---

After the next 124 blocks are read you'll see:
insert destination disk, press SPACE

18) You may now go back to 13) and continue the cycle of swapping disks back and forth as instructed. BE PATIENT! It may take 12 disk swaps. The last pass of reading blocks from the original source disk will probably not go all the way up to 124. It will probably end at some other number. That is your sign that this is the final swap. Put in the destination disk as instructed. After it writes the final blocks you are done. The program then lists the directory of the newly created disk.

APPENDIX B

HOW TO FORMAT A DISK

```
         Put new disk into drive 0:
COMAL:   PASS "n0:disk name,id"
BASIC:   OPEN 15,8,15,"n0:disk name,id"
         CLOSE 15
```

Replace "disk name" with any 16 character name for the disk. Replace "id" with anay two characters you wish to use as its ID. This ID should be unique from all other disk ID's you already have.

If you want to erase ALL programs and files from a disk you have already formatted and used, you can reformat the disk:

```
         Put disk to be reformatted into drive 0:
COMAL:   PASS "n0:disk name"
BASIC:   OPEN 15,8,15,"n0:disk name"
         CLOSE 15
```

Notice, it was the same as the format sequence EXCEPT you did not specify the ID.

COMAL USERS GROUP, U.S.A., LIMITED
5501 GROVELAND TERRACE
MADISON, WI 53716
(608) 222-4432

OTHER COMAL BOOKS OF INTEREST:

COMAL HANDBOOK by Len Lindsay
Spiral bound, 333 pages, $18.95
Matching disk (Commodore 4040 / 1541 format), $19.95
This is the reference book for CBM COMAL. It was written in cooperation with UniCOMAL, authors of the COMAL language for the PET, CBM, and Commodore 64. The book has already been translated into Swedish and Danish.

BEGINNING COMAL by Borge Christensen
Soft cover, bound, 333 pages, $19.95
Matching disk (Commodore 4040 / 1541 format), $19.95
This is an informal beginners tutorial, written by the founder of COMAL. It is designed to be used in a classroom. The book is imported from England. The book is also available in Danish and German.

FOUNDATIONS IN COMPUTER STUDIES WITH COMAL by John Kelly
Soft cover, bound, 313 pages, $19.95
Matching disk (Commodore 4040 / 1541 format), $19.95
This is a serious beginners tutorial, written to be used in a classroom. The book is imported from Ireland.

STRUCTURED PROGRAMMING WITH COMAL by Roy Atherton
Soft cover, bound, 266 pages, $24.95
Matching disk (Commodore 4040 / 1541 format), $19.95
This is the intermediate / advanced tutorial, written by a well known educator. It is designed to be used in a classroom. The book is imported from England.

COMAL NEWSLETTER:

COMAL TODAY, bimonthly, $14.95 per year
Contributors include Borge Christensen, COMAL founder, Len Lindsay, author of COMAL HANDBOOK, UniCOMAL, authors of CBM COMAL, and many other educators and software authors. It's packed with programming tips, latest news, helpful articles, and short adventures of CAPTAIN COMAL. A matching DISK subscription is available, for $65.70 (only $10.95 per disk).

ISBN 0-928411-01-X