TABLE OF CONTENTS

PROGRAM LISTINGS

INTRODUCTORY COMMENTS

A relative file provides the computer owner with comparatively quick access to a specific record for information, or for updating of the information in the record. When a program is run which activates a file of information for that type of activity, there is generally no predetermined limit to the length of time the program is "On Line". This determination is usually made by the person running the computer, after the information and updating requirements are completed.

For home application, most information storage/retrieval needs can be met by using a relative file. Included are a Family Budget system, a Things To Do listing system, an Appointment Scheduling system, Name & Address systems, various inventory files, and other applications. For a small business, many applications for this kind of file can be conceived by the business man, from general accounting to unique systems. Those with small businesses will find almost infinite versatility in the Texas Instruments, Inc., TI-99/4A disk system.

For most applications, all it takes is a file and a program to maintain the file. Besides file maintenance, the program should include routines to print or display file changes, records in the file, and file totals accumulated from specific fields in each of the file records. With the Extended Basic Command Module, it is an easy task to link two or more programs to the same file and is a way to get around the program size limitations which exist. Really sophisticated small business computer systems are possible using this computer.

The programs on the disk you purchased from me will provide you with hands on experience when loaded into the computer and run. These programs are also discussed in the text, at least those areas of the program which are appropriate to the subject being covered by the text. All the programs on the disk can be used without change after you have finished with the information in this text.

The text is written with the assumption that you are as familiar with the format of the instructions used in the example programs as possible, but that your understanding of them is not complete. The necessary understanding of the instruction formats can be obtained by a review of the programming guide which comes with your computer. I am more concerned with the use of certain instructions than I am with an explanation of their formats.

Computer hardware required to use this training disk: The 99/4 or 99/4A computer, a monitor, a disk drive controller and one disk drive. The programs used as examples use the monitor, rather than a printer. If your interest is in business systems, there are differences between them and home applications. I will cover the differences and provide programming examples. However, the completed examples are for a home application, because home applications are less complex, have a wider audience, and are easier to understand. The functions written into those programs are used in business systems as well and should not be

---

ignored by those of you who are strictly interested in business programs. I will often give extreme examples, both simple and complex, on the assumption that with anything in between, you should be able to work out on your own. My objective is to help you understand how to develop and program your own computer systems using relative files. If you have a printer, and/or the additional memory unit, I know this information will help you develop systems using those components.

In the section of the text that follows, I explain a program on your disk which you can run. By using the different options available to you from the menu screen, you can get a good feel of how a file maintenance program works.

The next section of text explains how to write a program to create a relative file. A relative file is created containing X number of dummy records. A file's size is determined by the length of each record and the number of records in the file. Both of these factors are under the control of the program that creates the file. Files are very easy to create, if you know how. It can be done with six or more lines of code, so you should have no problems learning how.

Dummy files are usually created with null characters in alphabetic fields and with a zero in numeric fields. Another program will be required to read and update the records in the file. That would be the file update program. They are much more difficult to program. Consequently, the majority of the text will be devoted to various aspects of the file update program.

OVERVIEW

In the Program Listings section of this text is a listing of a program named RELTVTEST. It is a very short program - 56 lines of instructions and comments - but contains an ultra simple example of most of the things that a person would want a relative file program to do. A summary of the main program functions are noted in the following, along with the beginning and ending program line numbers.

LINE NUMBERS		
FROM	TO	FUNCTION OF THE INSTRUCTIONS
100	140	Establishes the initial value of some variables used later in the program.
150	230	Creates a relative file containing 151 records.
240	260	Displays on the monitor screen an explanation of the options available to the user. Commonly referred to as the "Menu Screen".
270	320	Requests the user to enter thru the keyboard one of the available options defined on the screen. Accepts that entry and branches to the part of the program that performs the requested function.
330	350	Routine to read and display the requested record information on the monitor screen.
360	490	Routine to change a file record, then to update the file record on the disk.
500	630	Reads the file records sequentially and accumulates a total on one of the fields under certain conditions. When finished, displays the accumulated total.
640	650	Routine to wind-up the processing. It closes and deletes the file created earlier. Normally you will do all that you can to preserve the file, but some design considerations may call for deleting a file after it has been used.

The example program has very little value except as a way to illustrate the various elements in a file maintenance program. It should be obvious though, that if a file maintenance program develops into a large and hard-to-analize program, the size and complexity of the program must be attributed to the complexity of the processing logic, rather than to the code needed to satisfy unique requirements of random file processing. Since the file create and maintenance program has value only as a teaching aid, the two programs, normally separate, were combined into one program.

Run the program and use each of the options available to you. Several conclusions can then be verified about random file systems, which will help you recognize when such a program would be an appropriate solution to one of your data processing problems:

- You will find that you have more or less instant access to any of

---

the records on the file by entering the record number for the one you want. If the record number is not known, then the computer might take several minutes to search the file for the desired record. To read a record into the computer from the disk file, the INPUT # statement must specify the record number. That number may come from the computer operator, or it may be obtained in other ways. All of the "other ways" are much slower, but may save time in the long run by reducing the manual effort required from the computer operator to determine the record number.

- You will get a good feel for how long it takes the computer to get a disk record and display it on the monitor screen.

- Random files can be processed by the computer program either randomly or sequentially (record number sequence). It will not be unusual for a program to contain the programming routines to do both types of processing.

- Because of the possibility of human errors when recording or entering a record number to change, it would be very easy to update the wrong record. Various editing and other types of programmed routines can be used to reduce this risk.

- Report information can be displayed on the monitor, listed on the printer, or both.

- When one of the functions has been completed by the computer, the program normally will display a menu screen and request the operator to enter the next function the computer is to perform. Normally, one of the options is to end processing. If the program contains more than one menu screen, each of the screens will contain an option to allow the operator to go from one menu screen to the other.

To run the program, go to your computer, insert the disk you have received from me, and enter the following:

```
OLD DSK1.RELTVTEST
```

After the program is loaded into the computer, you need to type in RUN and press the Enter key. To stop the program run, enter record number 9999. That not only stops processing, but deletes the file from the disk.

Each of the major program functions are discussed in detail in the following sections of this manual, so a detailed analysis of this program would be redundant. However, I suggest that you turn to the program listing and see if you can follow the logic of the program instructions. If you run the program for a while, the programming instructions should be easier to understand. If you cannot understand what the program is doing by reading the instructions, you need to go thru the rest of this course.

CREATING RELATIVE FILES

In the Overview Section, I presented a program which creates a file, then updates the file. In practical applications, these two functions are separate programs. Once the programs have been written and are functioning properly, there is no longer a need for the file create program, except as a way to create another blank file on another disk. However, the file update and maintenance program has to refer to file name, and must define the attributes of the file generated by the file create program. Also, the file read and write statements in the maintenance program(s) generally will mirror the field names used by the create program when the new file was generated. If one person is writing both programs, the logical program to start is with the file create program.

This section will discuss ways to create files suitable for both home and business applications. If you are interested in developing business systems, I give some programming examples, but the complete programs used for illustration are more for home use. The difference between the two is that business systems are more complex. The programming requirements for business systems are more involved. For example, multiple record format files are the rule, rather than the exception for business systems. Creating files with multiple record formats is possible with this computer. There are some programming differences which I will illustrate later in this section. The upper limits on program size and on disk space are factors that have to be considered, but they can more or less be ignored as factors that will prevent including program functions that are necessary for the system. Along that same line of thought, the number of records a disk can hold is limited, but the file can be expanded to include two or more disks.

For home applications, single file, single record formats are probably most frequently used. These systems are relatively easier to define and program. They are easier to illustrate and to understand from a student's point of view. There are some differences between the two types of systems. But once the differences are understood, what is said about home applications can be applied to more complex systems.

PRELIMINARY PLANNING STEPS

Files contain records and records contain fields. Fields contain information of two types, numeric or alpha/numeric. The two types require separate fields, because they are referenced by two types of variable names. Fields need to be considered distinct from one another because they contain information pertaining to different subjects. This is the extent that data is segregated in files. The length of the information in a field may vary with respect to the same field in the other records. The only restriction which exists is the total number of characters that one record may hold.

It will be necessary to do some preliminary planning before sitting down and writing the program instructions to create the file you wish to use. Once the file has been created, you cannot go back and insert another field or two that you had forgotten about. You can go back and create another file containing the missing fields, but the file would contain dummy records. The information would have to be transferred to the new file, by another program. These, then, are the planning steps that I would recommend you follow, before writing the file create program:

1. Determine how many records your file should contain.

2. Establish the fields of information that each record is to contain. I will provide you with examples later that should help you determine what fields are.

3. Estimate the maximum length of information each of the fields will hold. If the field is to hold numeric information, you have no choice in the matter. All numeric information is carried in an 8 digit field, with one more character required for use by the computer's operating system. For example, if you want to reserve a field in the record for a 1 digit numeric code, it still will occupy 9 characters in the record. Alpha/numeric fields are 1 character longer than the information that is there, so they vary in size from one record to the next. The computer makes up the difference between what is actually there and the total record length by padding the record with characters.

   The objective here is to determine how long each record needs to be. Each record has to be the same length, but the length of the alpha/numeric fields can and does vary within each record. The record has to be able to accomodate the maximum length of the information to be entered into each of the fields.

4. Establish your file record length from the maximum field length requirements, then add one character per field required by the operating system, for either numeric or alpha/numeric fields.

5. Assign variable name codes to each field on your list. These variable field name codes should be short to reduce program entry time. The PRINT # statement limit of 112 characters is the major restriction on field name length. Once the program is up and running, the variable names used will not be of interest to anyone other than the programmer who has to look at the program listing to insert program changes. The name codes have to be unique - no duplicates.

PLANNING EXAMPLES

A HOME APPLICATION

For demonstration purposes, we will create a file to hold information on friends and relatives. The information to be stored will be name, address, phone number, birthdate and an extra field for miscellaneous

---

information. At the first of each month I want to go to the file and obtain a list of all who will be having birthdays that month. The file will also be used to look up the mailing address of those to whom I want to write.

1. Estimate the number of records needed.

I'm convinced that a file containing 150 records will provide plenty of space for my needs. As friends come and go, information on a record can be deleted and new information stored in its place. By delete, I mean the information in a record is deleted, not the record itself.

2. Identify record fields

A status code will be used to distinguish between active and inactive records. One field each is required for name, street, city, a second name line, telephone number, and birthday. A birthday month field is necessary to select those having a birthday in any given month.

3. Determine field length.

Since it is necessary to come up with a total record length, list the field names in one column, and set up another column to list the length in number of characters considered necessary.

FIELD NAME	SIZE	COMMENTS
Record Status Code	1	Use A for active, I for inactive
Full Name	25	
Optional Name line	20	Wife/Husband's name, or children
Street No., or P O Box	20	
City, State & Zip Code	25	Should all fit
Phone number	20	Make it an alpha/numeric field.
Birthday month	2	Make it an alpha/numeric field.
Birthday	15	Alpha/numeric too

Note that record number is not one of the fields that must be in the record. This means that the computer's operating system finds any record specified in the PRINT #, or INPUT # statements by counting records, starting with record number 0. It also means that we have to live with that kind of record numbering scheme.

4. Determine record length.

By adding the number of characters for each field, data requirements total out to 128 characters. The operating system also needs 1 character per field, or 8 more characters. That totals out to a record length of 136 characters (usually referred to as bytes). In the file create program it is necessary to specify the record length.

Once the record length has been determined, multiply that number times the number of records wanted in your file. If the result exceeds the number of characters your disk will hold, then consider making it a 2 disk system. A 40 track, single density disk will hold a least 358 records, as there are 358 sectors on a disk, each 256 characters long. The computer will not place a record on the disk so as to span two sectors. Any record length which exceeds half a sector will occupy one sector. With lengths of less than half a sector, the computer can place two or more records in a sector.

OTHER PLANNING CONSIDERATIONS

Once a file has been defined and you start using the file update program, then it becomes a relatively inflexible file. You will not be able to add new fields of information to the file. One way to allow for that type of expansion is to define one or more dummy fields which can be used if something needs to be added. I didn't do that in the example above. To add a dummy field, give it a field name and include it, for later use, in the INPUT and PRINT statements in the maintenance program. At a later date the maintenance program can be revised to include statements which use and maintain those reserved fields.

The need for more records is a problem which is simple to solve. Additional record space is easily obtained by taking another disk and running the file create program again. The file maintenance program will work just as well with the new disk file as it did with the previous one.

5. Assign variable code names to each field.

Variable codes are of two types. String variables (alpha/numeric) end with a $ sign. Numeric variables do not. Under number 3. above, eight fields were listed and they were all defined as string variables to reduce the space requirements. All that needs to be done is to assign codes to each field:

CODE	FILED NAME
P$	Status Code
Q$	Full Name
R$	Optional Information Line
S$	Street Address, or P O Box Number
T$	City, State, and Zip Code
U$	Phone Number
V$	Birthday Month
W$	Full Birthdate

PLANNING FOR BUSINESS SYSTEMS

My experience indicates that computer systems developed for a specific company are either conversions of the manual records system to a computer, or are conversions required to move the existing programs from obsolete equipment to a more advanced computer. While all businesses need certain types of accounting systems, most have developed one that

is more or less unique. Since very little originality is used when converting from a manual system to a computer, the problem of defining information fields for the computer system becomes much easier.
The previous example was a very simple record format. For a business system a more complex record set will be used. Many people feel that a payroll system is among the most difficult to develop. For the next example, let's define a payroll file. The example will contain 5 different record formats. There will be 5 record sets for each employee. You do have the option of either creating 5 different files, or one file with record sets. The programming will differ slightly to create a file with more than one record format. That is one of the reasons it is necessary to provide more than one example.

When determining file record field names, take a sheet of paper and make a list of the fields. Next, go through the list and record the maximum field lengths, as was done in the above example. One additional problem will be to assign each field to a particular record. If you will be using a lot of numeric fields, it is easy to get too many fields in one record. For example, in the PRINT # statements, there is a limit as to how many fields you can list. That limit is 112 characters for the whole statement, including line number, the instruction, the field names and the commas which separate the field names. All of the fields in the record have to be included in the PRINT # statement. This is a definate limiting factor, and needs to be considered in the planning phase when assigning fields to a record. As a general rule, 20 to 25 fields per record is a safe estimate. The example for a payroll system follows:

VARIABLE CODE	ESTIMATED SIZE	FIELD NAME
AA$	1	Status code
AB$	2	Employee No.
AC$	25	Full Name
AD$	20	Street Address
AE$	25	City, State, & Zip Code
AF$	15	Phone no.
AG$	12	Social Security no.
AH$	1	Shift code
AI$	1	Pay rate code
AA	8	Pay rate
AB	8	Sick leave accrual %
AC	8	Vacation pay accrual %
AD	8	Federal Withholding %
AE	8	State With. %
AF	8	Company retirement accrual %
AG	8	Reserved for later use

Total: 151 Characters, plus 16 fields = 167 character record length for rcd. #1.

Last Payday Record

AH	8	Hours worked (for hourly paid employees)
AI	8	O/T hours paid
AJ	8	Vacation $ paid

AK	8	Sick leave hours paid
AL	8	Gross pay
AM	8	Fed. With.
AN	8	State With.
AO	8	FICA With.
AJ$	2	Deduction type code
AP	8	Deduction amount
AK$	2	Deduction type code
AQ	8	Deduction amount
AL$	2	Deduction type code
AR	8	Deduction amount
AM$	2	Deduction type code
AS	8	Deduction amount
AN$	2	Pay adjustment type code
AT	8	Pay adjustment amount
AU	8	Net pay
AV	8	Balance sick leave earned
AW	8	Balance Vacation $ earned

YTD Totals

AX	8	YTD Gross earnings
AY	8	YTD Fed With.
AZ	8	YTD State With.

Total: 162 characters, plus 24 fields = 186 char. record length for rcd. # 2.

BA	8	YTD FICA With.

QTD Totals

BB	8	QTD Gross earnings
BC	8	QTD Fed. With.
BD	8	QTD State With.
BE	8	QTD FICA With.

Deductions Authorized

AO$	2	Type Code
BF	8	$ Amount per payday
AP$	2	Type Code
BG	8	$ Amount per payday
AQ$	2	Type Code
BH	8	$ Amount per payday
AR$	2	Type Code
BI	8	$ Amount per payday
AS$	2	Type Code
BJ	8	$ Amount per payday
BK	8	Balance Due

Employee history, & misc. information

AT$	2	Federal - No. of exemptions
AU$	2	State - No. of exemptions
AV$	1	Marital status code
AW$	2	Education level code
BA$	2	Work skills 1 code
BB$	2	Work skills 2 code
BC$	2	Work skills 3 code
BD$	2	Work skills 4 code

Total: 113 characters, plus 24 fields = 137 character record length for rcd. #3.

BE$	3	Current job no.
BF$	8	Start date
BG$	3	2 Job no.
BH$	8	Start date
BI$	3	3 Job no.
BJ$	8	Start date
BK$	3	4 Job no.
BL$	8	Start date
Pay history		
BM$	1	1 Pay rate code
BL	8	1 Pay rate
BN$	8	1 Start date
BO$	1	2 Pay rate code
BM	8	2 Pay rate
BP$	8	2 Start date
BQ$	1	3 Pay rate code
BN	8	3 Pay rate
BR$	8	3 Start date
BS$	8	Hire date
BT$	8	Termination date
BU$	2	Termination reason code
BV$	8	Layoff date
BW$	2	Rehire evaluation code

Total: 123 characters, plus 22 fields = 155 character record length for rcd. #4.

CA$	8	Last work evaluation review date
CB$	2	Evaluation code - Last review
CC$	2	Group insurance plan code
CD$	8	Birthdate
CE$	2	Company retirement plan code
BO	8	Co. retirement YTD accrual
BP	8	Co. retirement Total accrual - all years
BQ	8	Co. retirement current payday accrual
CF$	4	Standard labor distribution charge no.
BR	8	Advances to be repaid on termination
CG$	3	Current department no.

Total: 61 characters, plus 11 fields = 72 character record length for rcd #5.

Record #2 requires 186 characters, and is the longest of the five. This means all five records must be 186 characters in length.

One more record will be added at the end of the file to hold information pertinent to the system, but not to individual records. With this information accessable to the operator, he can revise the values, rather than make a program change to accomplish the revision:

BS	8	FICA Rate
BT	8	FICA Maximum
CH$	22	Dept. No. and Name
CI$	22	Dept. No. and Name
CJ$	22	Dept. No. and Name

CK$	22	Dept. No. and Name
CL$	22	Dept. No. and Name
CM$	22	Dept. No. and Name
CN$	22	Dept. No. and Name

THE PROGRAM

The computer does not keep track of variable names assigned to record fields. It does know the data type a field contains (numeric or string data). It would be possible to use different variable field names in each INPUT # and PRINT # statement in the program. A general rule to follow is to use the same variable field names to minimize the memory space required to store the variable information. Other programs which use the same file can use other field names. The information stored on a disk does not include field name assignments.

As a general rule, file create programs are short, while the file maintenance programs tend to exceed the limits of the computer's memory. The file create program is a good place to list the variable names assigned to each of the fields. Realistic variable names are 1, 2, or 3 digit alphabetic codes used in lieu of the longer, but more descriptive field names. These codes have to be as short as possible to allow all the fields in a record to fit in the PRINT # instruction. In fact, that is the next step.

It is important to use REM statements in the program to identify the code/field name assignments. This is because one or two months after the program is up and running well, the variable code assignments will have been forgotten. Later program revisions become almost impossible if that information is not available. The example file create program on the disk you received from me is the program to create the friends and relatives file. Rather than use REM statements, I assigned field names to the variables. See if you can anticipate what that does to the file which is created when the program is run.

The file create program for the friends and relatives file is so short it can be listed here in its entirety without any problem. You can load and run it anytime you wish. The maintenance program has an option to delete the file. If you do not wish to keep the file, you can use the update program (NUPDATE) to delete it from your disk. The first programming example is a copy of the friends and relatives file create program. Following that will be the instructions to create the payroll file:

```
 80 PRINT "PROGRAM NAME = NAMECREATE"
 90 PRINT "COPYRIGHT 1982, J. H. HARVEY"
100 REM SET VARIABLES USED IN THE INACTIVE RECORDS
110 P$="I"
120 Q$="NAME FIELD"
130 R$="OPTIONAL NAME LINE"
140 S$="STREET ADDRESS"
150 T$="CITY, STATE & ZIP CODE"
```

```
160 U$="PHONE NO."
170 V$="MO"
180 W$="FULL BIRTHDATE"
190 OPEN #1:"DSK1.NAME&A",RELATIVE,INTERNAL,FIXED 136
```

The "DSK1. part is always used, but what follows (NAME&A") is whatever name you want to give to the file. That name must appear in the Open statement in any program that wishes to access the file.

```
200 FOR I=0 TO 150
210 PRINT #1,REC I:P$,Q$,R$,S$,T$,U$,V$,W$
220 NEXT I
```

These 3 instructions create 151 inactive records in the file. The FOR - NEXT Loop is used to increment the record number variable I in the PRINT instruction. The information written into each field when the records are created is whatever the value of the variables are at the time. In this case, their values were set in lines 110 thru 170. If those lines had been missing, their values would have been "" (a null character) for $ variables and 0 for numeric variables.

```
230 PRINT "FILE CREATE FUNCTION COMPLETED"
240 PRINT
250 PRINT "TO MAINTAIN AND USE THIS FILE LOAD AND RUN THE PROGRAM NAMED
    NUPDATE."
260 CLOSE #1
270 END
```

It takes the computer about 2 1/2 minutes to create the file. As you can see, these are not complicated programs to write. This is an example of how to create a single format record file. When creating files containing 2 or more record formats, programming instructions are a bit different. A separate PRINT # statement is required for each record format, due to the difference in variable names, and in the number of fields. In the second example, where we have 5 different record formats, the program listed below will create 40 sets of records. At the end of the file the program will add one more record to hold variables for system constants which need not be included in each employee's record. Assume that the variable code assignments to field names has already been done using REM statements:

```
1630 OPEN #1:"DSK1.PAYFILE",RELATIVE,INTERNAL,FIXED 186
1640 FOR H=1 TO 40
1650 PRINT #1,REC I:AA$,AB$,AC$,AD$,AE$,AF$,AG$,AH$,AI$,AA,AB,AC,AD,AE,
     AF,AG
1660 I=I+1
1670 PRINT #1,REC I:AH,AI,AJ,AK,AL,AM,AN,AO,AJ$,AP,AK$,AQ,AL$,AR,AM$,AS,
     AN$,AT,AU,AV,AW,AX,AY,AZ
1680 I=I+1
1690 PRINT #1,REC I:BA,BB,BC,BD,BE,AO$,BF,AP$,BG,AQ$,BH,AR$,BI,AS$,BJ,
     BK,AT$,AU$,AV$,AW$,BA$,BB$,BC$,BD$
1700 I=I+1
1710 PRINT #1,REC I:BE$,BF$,BG$,BH$,BI$,BJ$,BK$,BL$,BM$,BL,BN$,BO$,BM,
     BP$,BQ$,BN,BR$,BS$,BT$,BU$,BV$,BW$
1720 I=I+1
1730 PRINT #1,REC I:CA$,CB$,CC$,CD$,CE$,BO,BP,BQ,CF$,BR,CG$
1740 I=I+1
1750 NEXT H
1760 PRINT #1,REC I:BS,BT,CH$,CI$,CJ$,CK$,CL$,CM$,CN$
1770 PRINT "FILE CREATION FUNCTION COMPLETED."
1780 CLOSE #1
1790 END
```

The above program example was not recorded on your disk. I have no intention of writing the update program to use that file. However, I did enter the instructions into the computer in order to confirm that it would create the file.

The maintenance program for the second example should contain edits to assure that any record number specified by the operator ended with a 5 or a 0. The operator would then be limited to specifying the first record in the set for an employee. The maintenance program will then have no problems reading in all five records for an employee. I suspect that some of you are wondering how to edit for a 5 or 0 input from the operator. This can be done by dividing the record number entered, by 5. Assign the length of that result to a variable. Obtain the integer value of the same division result, to truncate any decimal remainder, using another variable. Now compare the length of the two variables. If their length is the same, the number entered by the operator ended in a 5 or 0.

FILE MAINTENANCE PROGRAM - PRELIMINARY FUNCTIONS

The file maintenance program can either be a single program or may be split up into any number of programs which work with the same file. The determining factor is the upper limit on memory space available to hold the program. Proceed on the assumption that it can all be done in one program. If you do get a "Memory Full" message before the program is complete, it will be necessary to either buy the additional memory unit, or split the menu functions into two programs. Deleting REM statements and using the CALL FILES(1) command explained in the Disk Memory System manual will also buy a little more space, but usually not enough.

The preliminary part of a program contains statements and routines which are to be executed only once while the program is running, and which can logically be placed in the first part of the program. To start the coding of a program, take care of the preliminary program functions first. Sit down at a desk with a pad of paper and start recording the instructions. Write in the line numbers in the same way you will have the computer assign them. Line numbers are necessary to complete the GOTO, GOSUB, and IF statements. Preliminary functions are easy to code. This way, if you develop and work out each of the menu functions one at a time time, you will get to the END statement with a minimum amount of wasted effort and thought.

The main menu screen is usually the first major function to follow the preliminary part of the program. The part of the program which begins with the first instruction in the program and ends with the first instruction in the menu screen display routine, is the area that this section of the text is to cover. The following types, or categories of instructions and routines usually will be included in the preliminary part of the program, if they are included at all:

PROGRAM FUNCTION AND NAME

This is optional, but if you routinely print out your programs and file them, use the REM statement to identify the program and to describe its functions in a general way. The program name will be the name required to load it into the computer from disk.

INITIALIZE THE CONSTANTS USED BY THE PROGRAM

Most programs use constants in one form or another. Listed are three ways to get them into the program:

1. Assign constant values to variables.

   Variables are used for constants, as well as for field names. For example, if your program will calculate State Sales Tax for certain types of transactions, and it will always be, say 5%, you can set a numeric variable, say T, as being equal to either .05, or 1.05, depending on your calculation formula. The instruction to set that variable would be: T=1.05, or T=.05. T will equal 0 if that variable code assignment is not in your program. After the RUN command executes, T will equal 0 until the instruction is executed which sets the value to 1.05, or to .05.

2. Store the constants in a file record, or records.

If the constant is subject to change beyond the control of the computer system, you can carry the constant in a file record field. The field can then be updated as often as necessary by the operator. In the previous example, if T=1.05, and the State decides to change their Sales Tax rate, it will take a program revision to change the value of T, if a program instruction is used to set the value of T. Before making the change, the programmer has to first figure out which of the variable codes is for the State tax rate. The use of REM statements to define those variable codes can reduce an otherwise time consuming process when program revisions are necessary.

In the previous section on File Creation, you were given a programmed example of a payroll file. One additional record was added to the end of the file to hold variable information that could be revised by the operator. FICA rate and the FICA maximum were two of the constants provided for in that record. The Payfile maintenance program will have to include a menu option to allow the operator to revise those fields. The changes can then be taken care of by the operator selecting that option from the menu screen to change the field value, rather than having to revise the program.

The preliminary part of the program will contain instructions to read a file record to get the value of the variable. The variable code was established when it was assigned to the record field. Therefore, just reading in the record in the preliminary part of the program, will set the variable with the right value. To do that, simply code in an INPUT statement that specifies the right record number.

3. Use of INPUT statements

Constants may be entered by the operator through the keyboard, using INPUT statements. This method is always used when the program has to have the date. Since that entry is only required once during the run, the routine to get it should be in the preliminary part of the program. An example: INPUT "ENTER TODAY'S DATE":D$ The string variable D$ will then equal whatever the operator types into the computer before pressing the Enter key.

COPYRIGHT STATEMENTS

If you routinely copyright your programs, PRINT statements to the monitor will display the notice. Since it needs to be done only once during the run, the instruction should be in the preliminary part of the program.

DIMENSION STATEMENTS

These too should only be processed once during a program run. An example of DIM is included in the COPYN&A program listing:

```
     10  DIM A$(150),C$(150),D$(150),E$(150),F$(150),G$(150),H$(150),
         I$(150)
```

The string arrays this statement initializes is used later in the program to read in a complete file of 151 records, and to write them out onto another disk. Due to memory limitations, this was done in three passes, but could possibly have been done with two.

If you plan to use numeric or string arrays in your program for a table look-up, or something similar, this statement is required to set the number of repetitions. DIM statements should appear in the preliminary section of the program.

SPECIAL INSTRUCTIONS TO THE OPERATOR

The preliminary part of the program is also the place to code special instructions, or reminders to the operator, which do not have to be repeated each time the menu screen is displayed.

PROVIDE THE ABILITY TO CHANGE DISKS

If the data file is on one disk, and the maintenance program on another, there is a need to code in a program execution halt before any file Open statement is executed. This allows the operator time to exchange disks. It is a very simple routine that can be done two ways:

1.
```
PRINT "INSERT THE FILE DISK.  PRESS ENTER WHEN READY."
INPUT C$
```

2.
```
INPUT "INSERT THE FILE DISK.  PRESS ENTER WHEN READY.":C$
```

The character the Enter key generates is non-numeric, so the variable used has to end with a $ symbol. These instructions effectively halt processing until the operator presses the Enter key.

THE FILE OPEN STATEMENT(S)

Files are usually opened at the beginning of a run, and closed just prior to executing the END statement. When using multiple disk files, it is possible to switch disks anytime the disk drive light is out without Closing and Opening the file. To test this, do the following: 1) Run the NAMECREATE program twice to create the same file on two disks. 2) Load and run the NUPDATE program. Use the menu option to display the file records. Switch disks several times to check it out. However, if you have reasons to change disks under program control, offer that as one of the menu options.

The file Open statement to use in the maintenance program has to be exactly the same as the one used in the file create program. Open statements are also used to initialize the printer and the other components the program will be using. The Open statement cannot be repeated until after a Close statement, so be careful to conform to that requirement. If you plan to leave the file open while the program is "on line", the Open statement should be in the preliminary part of the program. Its' position in the program is fairly critical. It should appear after the routine to get the operator to insert the file disk, and has to be executed before any of the file read instructions, including the reads to establish values for variables and reads to create tables from information carried in the file. Both of those routines will also be in the preliminary part of the program.

MEMORY TABLES

This is such an important subject that more than one page will be used to cover this aspect of file maintenance. Memory tables are essentially numeric or string arrays containing information stored in the computer's memory. The information in the arrays will be used by program routines as required to complete functions selected from the menu screen. Memory tables are almost always loaded from a file that contains the data which goes into the table. Loading the table can generally be accomplished by just reading the file records, if the array variable is incremented as the records are read.

The examples which follow will demonstrate how to build and use memory tables taken from a single record format file, then a more complex example will be given. The objective of both examples will be to create and use a memory table to convert stock numbers in an inventory system to the record number. Record numbers are not known, but the stock numbers are known, and are unique for each inventory item. The conversion is necessary in order for the computer to read in the right record to update. In both examples, the complete inventory file will be read into the computer from instructions located in the preliminary section of the program. This has to be done only once to get the information into the array, so from that point on, the conversions are accomplished by a memory search instead of a disk search. Conversions of this type will slow down transaction entry, but do save manual effort. The computer could accept transactions faster if the operator is able to enter the record number. The gain and the loss would best be balanced on a cost scale to determine which way to go with the program.

EXAMPLE #1 - USING A 1 DISK INVENTORY FILE WITH 1 RECORD FORMAT

A preliminary step is to roughly determine if the array will fit into the available memory space. To do this for string data, multiply the number of records times twice the length of the data in each record. The stock number looks like this: 23-2279. This is 14 characters times 501 records, or slightly over 7,000 bytes. Since there is a dash in the stock number, a string variable will be used for the array. To roughly calculate the space required for numeric arrays, assume each numeric field will require 8 characters of memory.

The inventory record field names and assigned variable codes are as follows:

AB$ = Record status code.
AC$ = Part number
AD$ = Stock number
AB = Quantity on order
AC = Reorder point
AD = Quantity balance
AE = Cost per unit

The program statement to read a file record looks like this:

```
INPUT #1,REC I:AB$;AC$,AD$,AB,AC,AD,AE
```

The programming requirements can be summarized as follows:

1. Dimension the array Variable. This is accomplished by the following program instruction:

```
60 DIM A$(500)
```

2. Read the file and build the array. The instructions have to follow the file Open statement. The routine to load the array belongs in the preliminary part of the program. The array may be loaded from the file using the following three instructions:

```
110  FOR I=0 To 500
120  INPUT #1,REC I:AB$,AC$,A$(I),AB,AC,AD,AE
130  NEXT I
```

   Comments on the above routine: Each time a record is read, the values in all the variables, except A$(I), are overlaid by the data in the new record. Since I is incremented each time the program goes thru the FOR/NEXT loop, it eventually creates 501 different A$ variables.

3. Write a subroutine to convert the stock number into the record number. If there is no matching stock number in the array, the program is to return a record number over 500. By programming the conversion routine as a subroutine, any of the menu options which accept a stock number in lieu of the record number can use the subroutine to accomplish the conversion. The subroutine will normally appear in the main body of the program, rather than in the preliminary section of the program. But to complete the example, the subroutime will be shown. Most likely the subroutine will be placed near the END statement:

```
710  REM SUBROUTINE TO CONVERT STOCK NO. INTO RECORD NO.
720  K=0
730  IF C$=A$(K) THEN 790
740  K=K+1
750  IF K>500 THEN 770
760  GOTO 730
770  I=600
780  GOTO 800
790  I=K
800  RETURN
```

Comments on the above routine: I is the record number and C$ is the stock number which has to be converted. K is the counter used to step thru the array in line 730. The value of K becomes the record number when there is a match of stock numbers.

STOCK NUMBER CONVERSION - MULTIPLE FORMAT RECORDS, MULTIPLE DISKS

The conversion programming requirements will be similar if a more complex file is used, but there will be enough differences in the program statements that you need to look at another example. To set the stage: All conditions are the same as with the simple inventory records system, except there are three records in the file for each stock number. The file will consist of 5 disks, each containing 151 inventory record sets. Memory space will not be a problem as the Expansion Memory unit is to be used. The system will be on line 80% of the time to accept transactions entered thru the computer's keyboard. The program will do a record number conversion for each transaction. Then write them to a transaction file for later updating of the file. The record number for new inventory items being added to the file will be established by another routine that we will not be concerned about here.

The programming requirements applicable to the table array are about the same as for the simple system, with the following differences:

1. The dimension instruction should set the array large enough to handle up to 5 disks of inventory records.
2. When the table is being generated, the exchange of disks should be under program control.
3. The conversion subroutine will have to return the disk number as well as the record number. If no matching stock number exists, convert disk number to 21 and record number to 999.

The dimension statement will look something like this:

```
 50 DIM A$(754)
```

The following routine will build the in-memory array:

```
210 FOR M=0 TO 150
220 INPUT #1,REC I:AB$,AC$,A$(N),AB,AC,AD,AE
230 I=I+3
240 N=N+1
250 Next M
260 I=0
270 INPUT "INSERT NEXT INV. FILE DISK IN NUMERIC SEQUENCE.  ENTER A 9 IF
    THERE ARE NO MORE INV. DISKS.":E$
280 IF E$="9" THEN 300
290 GOTO 210
300 N=N-1
```

Since the value of N is incremented by 1 each time a record is read, the A$(N) value is retained in memory (line 220). I is the record number variable used in the Input statement. It is incremented by 3 after each read. The Input statement reads every third record in the file starting with record 0. Each time another disk is inserted, it is necessary to start over at record # 0, which is the reason for the instruction at line 260. The variable N can later be used in the conversion subroutine to determine when to stop searching the array for a stock number match. The M FOR/NEXT loop makes sure that 151 records on each disk is read for the array.

Another variable array could be dimensioned to carry the disk number, but it consumes valuable memory space and, if used, could cut down on the maximun number of disks the system can handle. In this situation, it will be preferable to calculate the disk number. The subroutine in the simple example will be replaced with the following:

```
610 REM SUBROUTINE TO OBTAIN RECORD NO. AND DISK NO.
620 K=0
630 IF C$=A$(K) THEN 710
640 K=K+1
650 IF K>N THEN 670
660 GOTO 630
670 I=999
680 DISK=21
690 GOTO 810
700 REM CALCULATIONS
710 B=K/151
720 DISK=INT(B)+1
730 II=DISK-1
740 IF II=0 THEN 790
750 I=K-(II*151)
760 I=I*3
770 GOTO 810
780 REM II=0
790 DISK=1
800 I=K*3
810 RETURN
```

Conversions using subroutines similar to this should average 10 to 15 seconds each. The time required to load the array into memory will run about 4 to 5 minutes a disk. In a test run using an array containing 1,427 numbers, the program stepped thru the complete array in 37 seconds, looking for an invalid number. It would have taken over 20 minutes to do a disk search. The time savings is considerable.

Here, K is another numeric variable which is used to step through the stock numbers in the memory array. Its value is set to zero when the subroutine is entered (line 620). The stock number entered by the operator (C$) is compared to the first stock number in the array. If the two numbers do not match, the value of K is increased by 1 and the procedure repeated, except the next stock number in the array is used in the comparison.

The numeric variable N was incremented by 1 as the array was loaded into memory. Refer to line number 240 in the previous example. N's value is used to tell the subroutine when the search for a match has reached the end of the array, then a no match condition will exist. The routine starting with line 650 handles this condition by setting record number to 999 and the disk number to 21. The program routines which use the subroutine will check for those values and will notify the operator of the rejection.

The above example starts very much like the previous subroutine example, but does not end that way. Go to line 630 and assume the computer found a match when K had been incremented to 325. The disk number will be 3 and the record number will be 69. Keep in mind there are 3 records in every inventory record.

Arrays (or tables) are usually loaded in from another file. This information does not necessarily have to be taken from the main file. It can be a separate file established and maintained just to provide the information for the array. The programming to do this will be very similar to the example given. If information is read in from a separate file, you will want to include in the routine a file Close statement as soon as the array is complete, assuming the program has no further need to read from that file.

USER OPTIONS - THE MENU SCREENS

The purpose of the menu screen is to provide the Operator with a list of functions the program will perform. The functions are associated with codes, which when entered, will identify the function selected to the computer program. It is doubtful any two programs can be found (other than duplicated programs) which have the same menu screen. With so much diversification, this aspect of file maintenance programs will be covered in general terms only. Refer to the NUPDATE program listing, lines 120 through 340, for an example of a menu screen routine.

In going back over my own programs, the menu screen section of the majority, are composed of the following type of instructions, appearing in the order listed:

- A REM statement to note the start of the menu screen routine
- Print statements to note the option codes & function descriptions.
- An INPUT statement to accept the operator's selection.
- IF statements, one for each code, to determine where in the program to go to perform the requested function.
- An error message at the end of the IF statements to notify the operator that one of the valid codes was not received.
- A GOTO statement to display the menu screen again, because if the program gets to this instruction it means that an invalid code has been entered by the operator.

The above outlines the programming functions, but fails to address the contents of a menu screen. That subject will be next. Before any programming instructions are coded, the programmer needs to sit down and list the functions which the program has to perform. The functions which belong in the preliminary part of the program are separate from the others. Those which may be selected by the operator belong on the menu screen. Any remaining functions either belong in the preliminary part of the program, or are included in the end of processing routine. My experience has been that even the most complex programs will fit within that framework. It is one of many ways to approach the problem, and it works.

Most file update programs will provide the operator with the ability to perform the following functions:

- A way to add new records to the file. Since a file is created with a certain number of records and the number itself is fixed, the maintenance program needs the ability to change inactive records to active ones. This type of change will generally require edits in the program to assure enough information has been entered to satisfy the processing requirements of the other routines in the program. The programmer must decide which fields in the new record are optional and which are required. The decision is usually based on the information requirements of the other routines in the program. Refer to the NUPDATE program listing, lines 350 through 860, for an

---

example of an add routine.

- A routine to delete a record which has become obsolete. This generally means changing a record status code from active to inactive. Moreover, this function should include the routines to delete all of the information which exists in the obsolete record. This needs to be done so that a part of the old record does not become a part of the new record. It is an easy, but tedious, routine to code. Refer to the NUPDATE program, lines 870 thru 1090 for an example of a delete routine.

- Provide for an interim change of record status code, when necessary. Some business programs will be involved with updating files containing information which have reporting requirements that go beyond the change of status for the record. The reporting requirements may prevent the deletion of the information in the record. In such a situation, an interim status code may be used as a way to reflect the change of status. The interim code used can be any code other than the ones used to show active or inactive status.

- Another function common to most file maintenance programs is to display all or selected parts of any records which the operator wishes to see. This is a display only type option. Routines of this kind generally identify the fields by name, then display the information in the record. Refer to lines 1520 thru 1710 of the NUPDATE program listing for an example of this type of programming.

- The maintenance program should provide ways to revise all fields of information in the record. If there are a limited number of fields involved, the revisions can be taken care of as illustrated by the NUPDATE program. Refer to lines 1720 thru 2470 in the program listing. This aspect of the file maintenance programming is the most time consuming and difficult.

- Providing reports of various kinds are another function of the maintenance programs. The individual reports are usually tied to separate screen option codes. I advocate using the printer as little as possible. Using the monitor screen for reporting not only saves wear and tear on the printer, but can save processing time. It is well known that the computer can be tied up for long periods of time listing information on the printer. For reports which must be saved, consider storing the information on a disk and providing a program to display the information when needed.

- The End Processing function - This function should at least include a CLOSE and an END statement. The CLOSE statement will complete the posting of any file update records being held in the output buffer, before going on to the END statement. If you just turn off the equipmemt instead of using an end processing function, some file updates could be lost - whatever was in the output buffer at the time the computer was turned off will be lost.

- The following is applicable only to business systems which are used so much that one or more computers are dedicated to one application. If one program is retained "On Line" for extended periods of time, consider providing a way for the computer operator to shut off the equipment for a brief period of time, with the ability to continue on where processing had stopped. To accomplish this requires the program to store certain variable values in a special disk record, and reading them back into memory later to restore the processing status. This will require the menu screen to give the operator two more option codes. One is used to store the value of the variables in the disk record. The other option resets the computer to its original status. More information on this approach is provided in the section of the text dealing with transactions.

FILE UPDATING FROM TRANSACTIONS

DEFINITIONS

File updating, from a programmers' point of view, is the function which reads in a file record, allows the operator to revise the information in one or more fields in the record, then writes the updated record to the disk file.

Transactions are the activity which trigger the file updating. For home applications, this activity is always less formally documented than it is for business systems. Although there are some exceptions, files in general contain information (fields) on a lot of items (records), and the information is in a constant state of change (dynamic). If the changes are not recorded by the computer, the information in the file very quickly becomes obsolete. When this happens, the file is worthless as a source of reliable information. These changes are generally referred to as transactions, regardless of the form they take.

General Comments

One of the primary functions of a file maintenance program is to provide the computer operator with a way to update the file using change transactions. My objective with this section is to cover the subject of transactions without becomming bogged down in a lot of detail.

GENERAL METHODS USED TO UPDATE FILE FECORDS

Assigning a code to each field

The easiest way to handle the update function is to assign a code to each field, read the record(s) into memory, have the operator enter the field code, then the change information, revise the field, and write the record back to the file when the operator is through with the changes for that record. This method is illustrated in the NUPDATE program listing, starting with line number 1720, and continuing to line number 2470.

This method works best for files which are contained on one floppy disk. If multiple disks are required to hold the file, it becomes less attractive due to a rise in the potential for human error, plus the extra manual work required of the operator to separate the transactions into 2 or more piles. But, for most home and simple business applications, this method should work very well.

Another general rule is that the file is updated before the computer displays the menu screen in preparation for the next transaction.

Transaction Forms

With the more complex business systems, two or more people will usually be recording and processing transaction activity which affects the same file. For example, with an inventory system, sales personnel will be writing sales tickets which represent sales of inventory items. Production personnel will be preparing stock requisitions for supplies and materials. Buyers will be issuing orders for low stock items. Shipping personnel should note actual stock withdrawals and back-ordered inventory already sold. Inventory personnel will be generating records of physical inventory counts. There will also be receiving reports issued when new purchases arrive. All this represents activity which must be posted to the file as change activity. As a general rule, the system is programmed on the premise that the information, or the forms will be routed to the computer operator for posting to the file records.

While preparing to create such a system, the programmer will look upon those forms as a way to segregate transaction types. It is also a good way to identify the fields of information which must be included in each type transaction. This is done to establish the format of the transaction to be used by the operator to update the file. Most of the forms will contain some information which is not maintained in the inventory file. This type of information does not have to be entered by the operator. Some companies will have their employees transcribe the information onto a keypunch form as a way to eliminate the unnecessary information. Since this requires additional manual effort, other companies have the operator enter the information directly from the form.

Transaction types differ. For example: What is the difference between a customer sales ticket and a stock requisition for material going to the shop? Both are inventory withdrawals. The difference is in the way the accountant records the transactions. The Credit may be to the same inventory general ledger account, but the corresponding Debit should differ. Consequently, the programmer of most business systems should be coordinating with the accountant, as a way to establish the accounting requirements, before finalizing the preliminary planning of the transaction types to use in the system. If the system is programmed correctly, reports from the system can save the Accounting Department a great deal of effort by providing totals to post to their ledgers. It is often possible for the computer to provide detail support for the totals, in the form of transaction files and/or listings.

If the volume of the transaction activity is high, the pressure on the computer operator is to get those transactions entered. In this kind of situation, the operator can enter more transactions if it is not necessary to read and write the file record for each transaction. The method used under those conditions is to accept the transactions as fast as they are entered, then store them in another file. The master file can be updated later using the transaction file.

Such a file update run idealy needs to be scheduled early enough to allow the operator the time to follow-up and correct transactions which are rejected during the run, before the end of the shift. The best way to handle rejections from a transaction file is to list the rejected transactions on the printer, in order for the computer to continue processing. The operator should not be expected to make the correction on the spot. Most of the time the operator will need to call someone else to determine the correct information. Those calls need to be placed before the people go home, or even worse, go off on vacation. Generally there will be no problem in handling the corrected entry in the next day's business, or entering them in with the next shift's transactions.

I think this computer can handle really complex business systems. The programming can be done, and I think the hardware can withstand being used for long periods of time, particularly if a good surge protector is on the power line between the outlet and the computer equipment. In my opinion a good surge protector is essential equipment. I've had my computer turned on 10, 11 hours at a time without any problems.

PROGRAMMING CONSIDERATIONS WHEN USING TRANSACTION FILES

There are two major functions involved, and they will be covered separately. One is the creation of the transactions in a disk file from change activity entered by the computer operator. The other function is updating of the master file from the the transaction file. As a general rule these two major functions will be accomplished by different programs, to allow as much space as possible for in-memory arrays.

Creating the Transaction File

Initially, the file is created the same as any other random file. That procedure has already been discussed in the file create section of this text, and another programming example is unnecessary. By way of an example, I will define a universal record format for a transaction file, one which can be used to satisfy most file updating requirements. This approach would not necessarily be the best, and is certainly not the only format for transaction file records.

To implement the scheme, a basic rule to follow is that for every field to be revised, the program is to generate one transaction record. Each of the master file record fields will be assigned a number. This number will be used in the transaction record to identify the field to change in the master file record.

Let's go ahead and define the transaction record fields. We will need the record number and the disk number, if the master file is on multiple disks. Another requirement is a code to tell the update program if the transaction has been posted. That would be our rerun provision. A transaction code field is required to identify the type of change the update program is to make. Last, but not least, we need a data field to hold the change-to information. Since numeric information can be assigned to a string variable, that is what will be used for a data field. Numeric string data can be converted to a numeric variable later as required.

Listed are the field names for the transaction records, their maximum length, and the variable codes assigned to each field:

V. Code	Field Name	Max. Length	Comments
PSTAT	Posted Status Code	8	One digit required
DISK	Master File Disk No.	8	2 digits required at most.
RCD	Master File Record No.	8	Four digits required
FLD	Field No. to Update	8	2 digits
T/C	Transaction Code	8	2 digits
DAT$	Change Data	25	Alpha/Numeric
	Total	65	
+ 1 character per field O. H. =		6	
	Total Record Length	71	Characters

The variable codes assigned to the fields are longer than usual, and more descriptive. There is no problem getting six variable names into a PRINT # statement. If the names are too long, it becomes a chore to use them in the program statements.

Transaction Files, Miscellaneous Considerations

1. Transaction codes need to be discussed further. This code has to indicate to the update program how numeric data is to affect the master file field. With string data, the only option available is to replace the old data with the new data. With numeric fields, the code must tell the update program if it is to add to, subtract from, or replace existing information. Only three codes are necessary to satisfy this minimum requirement. However, a more complex numbering scheme is generally used. The accountants and other management people want more detail to properly segregate accounting totals accumulated during processing runs. The complex numbering scheme is used to accomodate their reporting requirements.

2. What to do about new additions to the file - The computer operator will be entering some transactions which will revise more than one field in the master file. When those transactions are entered by the operator, the program generating the record for the transaction file will have to produce one record for each changed field. A new add transaction would conceivably change every field in a record. This requires the program to generate a multitude of transaction records. It is best to update the master file record directly for new adds, if possible.

3. After all transactions are entered, it is easy for the update program to determine when the end of the file has been reached if the unused records in the transaction file are filled with zeroes, or null characters.

4 Another consideration is to provide the operator with a way to turn off the computer for short periods. The system must be able to keep adding transactions to the end of the file when the operator returns.

This presents something of a problem to the programmer. To allow the computer program to continue processing as if no interruption had occurred, the value of some variables must be saved on disk before the program run terminates. Then the system must restore those values when the operator returns. Accomplishing this objective requires the Programmer to do three things:

- Include instructions in the file create program to produce a special record containing fields in which the variables can be stored. The programmer must plan this aspect of the update processing before writing the file create program. In the File Create section of this text, I provided a programming example for a payroll file which includes a special record added to the end of the file.
- Provide a menu screen option to allow the operator to shut off the computer temporarily. This routine saves the variable values on the special disk record, then closes the files before going to the End statement.
- Provide a menu screen option to allow the operator to return from a temporary shut off of the computer. This routine reads the special record to restore the variable values. With a normal start-up, the first transaction written to the transaction file goes to record number 0. On a return from a temporary break, the next transaction to process must be placed at the end of the file. One of the variables to save is the transaction file record number. Other variables to save are any of the total accumulations being performed by the program.

File Updating From the Transactions.

Updating a master file from a transaction file complicates the programming requirements and is an option which results in a master file that is not current most of the time. Whenever possible, it is better to update the master file as the change transactions are entered by the operator. However, if the master file is on 6 disks, the operator would rebel at having to change disks for every transaction that processes. Also, the potential for operator error would be very high.

The error I am referring to is the operator inserting the wrong disk. It is possible to program in a check of a disk's name to eliminate errors of this type. A less atractive alternative is to carry the stock number in the transaction record for comparison when the transaction is processed by the update program. The disk's name is on the 0 record of the disk's index file. The Disk Memory System manual explains how to check that record. I'll concede that it is possible to reduce that risk using a programmed routine to make that check. Even so, the time consumed by changing disks, then making the test of the disk's name, would probably slow down the updating function so much that updating from a transaction file is a more attractive alternative. The file update program will be programmed to make one pass through the transaction file for each disk, and in that way reduces the amount of disk swapping required of the operator. Circumstances can exist where the logical choice is to update the file from a transaction file.

If the field to be updated in a master file record is determined from a field number in the transaction record, it implies the update program will contain X number of field update routines - A separate routine for each field in the master file. This makes X equal the number of fields in the master file. The update program will select a transaction to process, determine the field affected from the information in the transaction record, and branch to the routine that updates that field.

Under ideal conditions, the computer system will consist of two disk drives, so that the master file disk is located in one drive, and the transaction file disk is in the other drive. If the equipment is limited to one disk drive, the transactions will have to be read in then stored in a memory array. If the volume of transactions is very high, this procedure might be repeated several times to process all of the transactions. The programming to accomplish this is illustrated in the COPYN&A program listing located in the last few pages of this text.

MISCELLANEOUS FUNCTIONS APPLICABLE TO TRANSACTIONS

EDITING

With business systems, I advocate that one person or department be responsible for the accuracy of each file. Someone other than the computer operator. If all transactions are funneled through that department, they have the opportunity to check the documents for propriety, completeness of information, readability of the information to be entered, and the opportunity to establish control totals on critical fields, where input errors cannot be tolerated. In most companies, every employee has more work than they can do. If no one has been assigned responsibility for the accuracy of a file, chances are good the files will be neglected.

People catch problems which the computer cannot detect. On the other hand, people reviews tend to vary in quality and may not be done at all if the pressures are great to do other work. The computer can detect some types of errors that a reviewer would not be expected to catch. Checks programmed into the computer can be counted upon by the managers to function reliably. However, certain types of errors cannot be detected by the computer. For instance, if the operator types in 97 instead of 79, a transposition error, the computer has no way of knowing what should hve been entered. Errors of this kind can be flagged as having occurred only if there is an item by item manual review, or by using control totals.

Checks the computer can make of the data it is to process will pay off by rejecting the data before it can affect the file and other output. It is a hundred times better to reject bad transactions than to try to reverse the effects of one that did process. The amount of effort expended on controls has to be tempered by the importance of the data and the result if errors did occur. The cost of manual controls should not be more than the controls are worth. It is best to concentrate on the more critical areas.

It is also helpful to know where errors will originate. The computer keyboard is one source of possible error. Some keyboards duplicate characters at random frequencies and typing errors are common. If data entry is based on hand written documents, errors will result from trying to interpret illegible writing.

It is generally up to the programmer to determine which edits to include in the program. The department responsible for the accuracy of the file should know what those edits are, and be given the opportunity to accept or reject proposals in that area. A program without edits to detect and reject data input errors is not an acceptable program, particularly business system programs.

Report Rejected Input Data

If data entered by the operator is rejected, for any reason, it has to be reported in order to correct and reprocess the transaction. The rejected transaction cannot just disappear. If the update is being accomplished by a dialogue between the computer and the operator, the rejection can be noted on the monitor screen, with the expectation that the operator will make the correction right away. However, if the transaction is one of many being processed from a file of transactions, it is best to print the rejection notice on the printer and have the computer program continue processing, using the next transaction.

Total Accumulations for Accounting

Nearly every business system is an opportunity to provide the Accounting Department with totals for posting to ledgers, and with the detail supporting the totals. A good example is the inventory adjustments resulting from physical inventory counts. If the accountants record the adjustments, the difference between the count and the inventory records is important to them. When recording the count, the computer can accumulate this difference in terms of quantity and dollars. It also has the ability to produce a file containing just these differences as a way to support the posting total. If this file can be listed on the monitor, it may not be necessary to print the file. Other routines can be added to the program to display significant adjustments which could trigger a recount.

Disks containing transactions are dangerous things to leave lying around. The danger is that the transactions may be processed more than once. If it is not necessary to keep the file after the transactions process, I suggest the following programming approach:

1. Have the program which writes the transactions to the file, create

the file as a new file each time the program runs. This is logically done in the preliminary portion of the program. Next, write the transactions to the file, closing the file normally when the session is over.

2. The program which reads the transactions for the file updating function should close and then delete the file when finished. This ensures that the transactions will not be posted twice.

If it is necessary to retain the transaction file, the programmer will have to provide some way to prevent the file from being posted twice. One way is to reserve a particular record in the transaction file for use as an indicator. The indicator can tell the update program that the file has already been posted. This brings up another important programming consideration - electric power failure.

Sooner or later this is going to happen to you, and a well designed system will provide for that occasion. The electricity doesn't have to be off very long before the program in memory is erased. Assuming there is no disk or equipment damage, the file update program will have to be reloaded and the transaction updates continued. The file as it stands has been updated by some transactions and not by others. If the update is from a transaction file, then how does the computer or the operator know where or how to restart processing? In that situation, the program has to be able to pick up where it left off. If you have a copy of the file as it was before the update, then you are in good shape. Simply rerun the program, being sure to use the back-up disk file. Another way to do it without having to have a back-up disk file is to update the transaction record with an indicator as each transaction is processed. This approach requires a two disk drive computer system, but saves the time it takes to back-up the file after every update. It also calls for routines in the update program to test the indicator and not use the transactions that have already been posted. I have mentioned two ways to provide for a rerun, and you can probably think of other ways. The important thing is to design the system to include the rerun capability, if transaction files are used to update the master file.

FILE CONTROLS

File controls are really a management tool to check on the integrity of the system. Control totals are usually used to report the status of the check. If the control totals are out of balance, they are not much help in locating the problem. They more or less indicate the extent of the problem.' Outside audit firms are required to point out the absense of file controls. I can tell you how to use totals for file controls. The decision to included them in a program depends a lot on the information which will be in the file.

Input Transaction File Control Totals

Let's go back to an inventory system as an example. Assume that the department given the responsibility for the accuracy of the file decides they want the computer to print out a report total of number of transactions it got by transaction type and a total on the quantity field in those transactions. If the department also had access to the transaction file created for the file update, it is possible to resolve any differences between their totals and the totals processed by the computer. This is an effective control over the transactions that the computer processes. It would be very difficult for anyone to slip in an unauthorized transaction without some one knowing about it. If the controls are checked and out of balance conditions resolved before the update run, the control is even better. Such a control will also catch input typing errors that affect the quantity field.

Master File Control Totals

The controls are very similar to the controls used on transactions. They generally consist of two totals. One total is the number of active records, and the other is a total on one of the file record fields. The quantity field again will be used as an example. There will be a record in the file reserved for control totals. One total is for the number of active records in the file as of the end of the last file update run. The other total is a sum of the inventory quantity balance in each record. Those fields will provide the beginning balance at the start of the next file update run. The control record also provides fields for additions and deletions from transaction update activity. These totals are accumulated during the update run by the update program and posted to the control record at the end of the processing run. They can also be posted as necessary if the transaction file approach is not used. After they have been posted, it is possible to calculate how many active records should be in the file and what the new quantity balance ought to be. To find out if that is the case, a sequential pass through the file is necessary. A printed report will be produced at the end of the sequential run, the transaction activity totals deleted, and the new balances posted to the control record. The report should go to a manager. If there is an out of balance condition indicated, the problem will have to be investigated and corrected.

Testing

Programs need to be tested to be sure they will run and produce the right results. It is relatively easy to find out if the program will run. Determining if the program is producing the right results takes time. The updated file records, transactions files and reports need to be checked, using fictitious transactions, to test the accuracy of the processing routines in the program. During the tests, every type of transaction the system is designed to process should be used. Testing has to be a mix of valid and invalid data. If the edits are good enough, they will reject the invalid transactions. The proper mix is about 5 to 10 invalid transactions to every good transaction. Reporting accuracy must be confirmed through manual calculations. Each total accumulation and detail line type needs to be verified as accurate. The file create program may be used to provide a file free of tests results, after the testing phase is finished.

DOCUMENTATION

HOME APPLICATIONS - SELF PROGRAMMED

The documentation for these programs are usually minimal. One suggested method is to take a printed copy of the final version of the program and paste it onto note paper, then file it in a note-book. On the first page, record the name of the program used to load it into the computer. The disk the program is stored on is also good information to keep. Behind the printed program listing I will file an explanation of variable codes used in the program. This would be the extent of my documentation. It provides enough information to run the program, or to figure out any program changes to make.

HOME APPLICATIONS - COMMERCIAL

If you are selling your programs to other computer owners, it is an entirely different situation. The primary objective is for the written description of the program to contain sufficient detail to enable the customer to use it. A secondary objective may be to provide customers with the type of information which will allow tailoring the program to individual preferences.

The bulk of the documentation will be devoted to explaining in detail to customers how to load and use the program. The first page will probably be an index where each of the functions are listed. Next, include a detailed explanation of those functions. If any of the functions have been written in such a way as to allow the program to abort under certain conditions, consider including a description of how the customer is to get back to where they were before the abort took place.

The instructions may include a write-up covering why and how to make back-up copies of the program and data files.

OVERVIEW PROGRAM

PROGRAM NAME: RELTVTEST

```
100 Q$=""
110 L=1
120 M=50
130 N=100
140 O=150
150 OPEN #1:"DSK1.CUSTOM",RELATIVE,INTERNAL,FIXED 60
160 FOR I=0 TO 150
170 L=L+1
180 M=M+1
190 N=N+1
200 O=O+1
210 PRINT #1,REC I:Q$,L,M,N,O
220 NEXT I
230 RESTORE #1
240 PRINT "ENTER RECORD NO. YOU WANT DISPLAYED FROM 0-150"
250 PRINT
260 PRINT "ENTER 999 TO CHANGE A RCD.  ENTER 9999 TO STOP PROGRAM.
    ENTER 888 TO READ FILE SEQ."
270 PRINT
280 INPUT I
290 IF I=9999 THEN 640
300 IF I=999 THEN 360
310 IF I=888 THEN 500
320 IF I>150 THEN 240
330 INPUT #1,REC I:Q$,L,M,N,O
340 PRINT "Q$ = ";Q$:"L = ";L:"M = ";M:"N = ";N:"O = ";O
350 GOTO 240
360 REM CHANGE ROUTINE
370 PRINT "ENTER RCD # - USE 999 TO GO BACK TO DISPLAY RECORDS."
380 REM ACCEPT ENTRY FROM KEYBOARD
390 INPUT I
400 IF I=999 THEN 240
410 IF I>150 THEN 370
420 Q$="RCD UPDATED"
430 L=L+1
440 M=M+1
450 N=N+1
460 O=O+1
470 PRINT "RECORD ";I;", ";Q$;", ";L;", ";M;", ";N;", ";O
480 PRINT #1,REC I:Q$,L,M,N,O
490 GOTO 370
500 REM READ FILE SEQUENTIALLY & ACCUM L FROM CHANGED RCDS ONLY.
510 RESTORE #1
520 I=REC(1)
530 INPUT #1,REC I:Q$,L,M,N,O
540 PRINT "PROCESSING RCD # ";I
550 IF I=150 THEN 590
560 IF Q$="" THEN 520
```

```
570 LL=LL+L
580 GOTO 520
590 IF Q$="" THEN 610
600 LL=LL+L
610 PRINT "ACCUM OF THE L FIELD FOR ALL ACTIVE RECORDS IS ";LL
620 RESTORE #1
630 GOTO 240
640 CLOSE #1:DELETE
650 END
```

The following program uses and maintaines a Name and Address file created by the NAMECREATE program used as an example program in the file create section of the text.

The name of this program is NUPDATE, and the file it uses is named NAME&A.

```
90    REM PROGRAM NAME IS NUPDATE
100   PRINT "COPYRIGHT 1982, J. H. HARVEY"
110   OPEN #1:"DSK1.NAME&A",RELATIVE,INTERNAL,FIXED 136
120   REM MAIN MENU SELECTION SCREEN
130   PRINT
140   PRINT "ENTER THE OPTION CODE FOR THE FUNCTION YOU WANT TO USE."
150   PRINT
160   PRINT "1 = ADD A NEW NAME AND ADDRESS TO THE FILE."
170   PRINT "2 = DELETE AN EXISTING NAME & ADDRESS."
180   PRINT "3 = LIST UNUSED RECORD NO'S AVAILABLE"
190   PRINT "4 = LIST EVERY ONE HAVING A BIRTHDAY THIS MONTH."
200   PRINT "5 = DISPLAY REQUESTED RECORDS"
210   PRINT "6 = REVISE EXISTING NAME AND ADDRESS RECORD."
220   PRINT "9 = END PROCESSING AND SAVE THE FILE."
230   PRINT "98= END PROCESSING AND DELETE THE FILE."
240   INPUT A
250   IF A=1 THEN 350
260   IF A=2 THEN 870
270   IF A=3 THEN 1100
280   IF A=4 THEN 1270
290   IF A=5 THEN 1520
300   IF A=6 THEN 1720
310   IF A=9 THEN 2480
320   IF A=98 THEN 2500
330   PRINT "OPTION CODE ";A;" INVALID,  TRY AGAIN."
340   GOTO 130
350   REM FILE ADDITIONS ROUTINE
360   PRINT
370   PRINT "ENTER RECORD NUMBER TO USE FOR NEW ADD."
380   PRINT "OR, ENTER 999 TO RETURN TO THE MAIN MENU SCREEN."
390   PRINT
400   INPUT I
410   IF I=999 THEN 130
420   IF I>150 THEN 510
430   IF I<0 THEN 510
440   INPUT #1, REC I:P$,Q$,R$,S$,T$,U$,V$,W$
450   IF P$="A" THEN 470
460   GOTO 550
470   PRINT
480   PRINT "RECORD IS ACTIVE, SO CANNOT BE USED."
490   PRINT "NEED AN INACTIVE RECORD NO."
500   GOTO 360
510   PRINT
```

```
520    PRINT "RECORD NO. ";I;" IS OUT OF THE VALID RANGE."
530    PRINT "TRY AGAIN."
540    GOTO 360
550    P$="A"
560    INPUT "ENTER NAME":Q$
570    IF Q$="" THEN 560
580    INPUT "OPTIONAL: ENTER BUSINESS NAME, OR PRESS THE ENTER KEY TO
       CONTINUE.":R$
590    INPUT "STREET ADDRESS OR P O BOX NO.":S$
600    INPUT "NOW CITY, STATE AND ZIP CODE":T$
610    INPUT "PHONE NO., OR IF NOT KNOWN PRESS THE ENTER KEY TO
       CONTINUE.":U$
620    INPUT "BIRTHDAY MO. ONLY AS A NO. FROM 1 THRU 12, OR PRESS ENTER
       TO CONTINUE.":V$
630    IF V$="" THEN 730
640    C=ASC(V$)
650    IF C>57 THEN 680
660    IF C<49 THEN 680
670    GOTO 710
680    PRINT "MONTH OF ";V$;" IS OUT OF VALID RANGE.  TRY AGAIN."
690    PRINT
700    GOTO 620
710    C=LEN(V$)
720    IF C>2 THEN 680
730    INPUT "ENTER FULL BIRTHDAY, OR PRESS ENTER TO CONTINUE.":W$
740    X$=P$&Q$&R$&S$&T$&U$&V$&W$
750    B=LEN(X$)
760    IF B>128 THEN 810
770    PRINT
780    PRINT "RECORD NO. ";I;" DISK RECORD UPDATED TO ADD ";Q$
790    PRINT #1,REC I:P$,Q$,R$,S$,T$,U$,V$,W$
800    GOTO 360
810    E=B-128
820    PRINT "RECORD IS TOO LONG BY ";E;" CHARACTERS."
830    PRINT "DISK RECORD WAS NOT ADDED."
840    PRINT "RECORD LENGTH CANNOT EXCEED 128 CHARACTERS."
850    PRINT "TRY AGAIN."
860    GOTO 360
870    REM DELETE PROCEDURE
880    PRINT
890    INPUT "ENTER RECORD NO. TO DELETE FROM 0 THRU 150. ":I
900    IF I>150 THEN 970
910    IF I<0 THEN 970
920    INPUT #1,REC I:P$,Q$,R$,S$,T$,U$,V$,W$
930    IF P$="A" THEN 990
940    PRINT "RECORD ALREADY DELETED."
950    PRINT "NO ACTION TAKEN"
```

```
960   GOTO 130
970   PRINT "RECORD NO. ";I;" OUT OF THE VALID RANGE."
980   GOTO 950
990   P$="I"
1000  R$=""
1010  S$=""
1020  T$=""
1030  U$=""
1040  V$=""
1050  W$=""
1060  Q$="DELETED"
1070  PRINT #1,REC I:P$,Q$,R$,S$,T$,U$,V$,W$
1080  PRINT "RECORD NO. ";I;" DISK RECORD DELETED."
1090  GOTO 130
1100  REM LIST UNUSED RECORDS
1110  F=0
1120  G=0
1130  FOR I=0 to 150
1140  INPUT #1,REC I:P$,Q$,R$,S$,T$,U$,V$,W$
1150  IF P$="A" THEN 1230
1160  PRINT "RECORD NO. AVAILABLE ";I
1170  F=F+1
1180  G=G+1
1190  IF G<15 THEN 1230
1200  INPUT "PRESS ENTER KEY WHEN READY FOR MORE.":Z$
1210  G=0
1220  PRINT
1230  NEXT I
1240  PRINT "TOTAL OF ";F;" RECORDS AVAILABLE."
1250  INPUT "PRESS ENTER KEY WHEN FINISHED.":Z$
1260  GOTO 130
1270  REM ROUTINE TO LIST ALL WITH BIRTHDAYS THIS MONTH.
1280  INPUT "ENTER THE MONTH AS A NO. FROM 1 TO 12.":G
1290  IF G>12 THEN 1500
1300  IF G<1 THEN 1500
1310  FOR I=0 TO 150
1320  INPUT #1,REC I:P$,Q$,R$,S$,T$,U$,V$,W$
1330  IF P$="A" THEN 1350
1340  GOTO 1450
1350  REM IF ACTIVE CK. FOR A MATCH ON MONTH
1360  Y$=STR$(G)
1370  IF Y$=V$ THEN 1390
1380  GOTO 1450
1390  PRINT "RECORD NO. ";I;" ";Q$;" BIRTHDAY IS ";W$
1400  F=F+1
1410  IF F>10 THEN 1430
1420  GOTO 1450
```

```
1430  F=0
1440  INPUT "PRESS THE ENTER KEY TO CONTINUE ":Z$
1450  NEXT I
1460  PRINT
1470  PRINT "SEARCH OF THE FILE FINISHED"
1480  INPUT "PRESS THE ENTER KEY WHEN FINISHED":Z$
1490  GOTO 130
1500  PRINT "MONTH OUT OF RANGE ";G
1510  GOTO 1280
1520  REM DISPLAY REQUESTED RECORDS
1530  PRINT "ENTER RECORD NUMBER FROM 0 TO 150, OR ENTER 999 TO RETURN
      TO THE MAIN MENU."
1540  INPUT I
1550  IF I=999 THEN 130
1560  IF I<0 THEN 1700
1570  IF I>150 THEN 1700
1580  INPUT #1,REC I:P$,Q$,R$,S$,T$,U$,V$,W$
1590  PRINT "THIS IS RECORD NO. ";I
1600  PRINT "RECORD STATUS CODE = ";P$
1610  PRINT "NAME ";Q$
1620  PRINT "OPTIONAL NAME ";R$
1630  PRINT "ADDRESS ";S$
1640  PRINT "CITY, ETC ";T$
1650  PRINT "PHONE ";U$
1660  PRINT "BIRTH MONTH ";V$
1670  PRINT "BIRTHDAY ";W$
1680  PRINT
1690  GOTO 1530
1700  PRINT "RECORD NO.";I;" NOT IN THE VALID RANGE."
1710  GOTO 1530
1720  REM ROUTINE TO REVISE AN EXISTING RECORD
1730  PRINT "MAKE A NOTE OF THESE CHANGE CODES."
1740  PRINT "1 = RECORD STATUS CODE"
1750  PRINT "2 = CHANGE NAME"
1760  PRINT "3 = CHANGE THE OPTIONAL NAME LINE"
1770  PRINT "4 = STREET ADDRESS"
1780  PRINT "5 = CHANGE CITY, STATE AND ZIP CODE"
1790  PRINT "6 = TO CHANGE PHONE NO."
1800  PRINT "7 = BIRTHDAY MO. CODE"
1810  PRINT "8 = CHANGE BIRTHDAY DATE"
1820  PRINT "9 = HAVE THE COMPUTER UPDATE THE DISK FILE AND RETURN TO
      THE MAIN MENU."
1830  PRINT
1840  PRINT "CHANGES ARE MADE TO EACH FIELD BY REPLACING WHAT WAS THERE
      WITH WHAT YOU ENTER."
1850  INPUT "ENTER RECORD NO. YOU WISH TO CHANGE":I
1860  IF I>150 THEN 1890
```

```
1870 IF I<0 THEN 1890
1880 GOTO 1910
1890 PRINT "RECORD NO. ";I;" OUT OF VALID RANGE."
1900 GOTO 1850
1910 INPUT #1,REC I:P$,Q$,R$,S$,T$,U$,V$,W$
1920 PRINT "RECORD NAME NOW IS: ";Q$
1930 U=0
1940 INPUT "ENTER CHANGE CODE ";C
1950 IF C=9 THEN 2350
1960 IF C>8 THEN 1990
1970 IF C<1 THEN 1990
1980 GOTO 2010
1990 PRINT "CODE ";C;" OUT OF RANGE."
2000 GOTO 1940
2010 INPUT "ENTER REVISED INFORMATION ":C$
2020 U=1
2030 IF C=1 THEN 2110
2040 IF C=2 THEN 2140
2050 IF C=3 THEN 2170
2060 IF C=4 THEN 2200
2070 IF C=5 THEN 2230
2080 IF C=6 THEN 2260
2090 IF C=7 THEN 2290
2100 IF C=8 THEN 2320
2110 PRINT "STATUS CODE WAS ";P$;"NOW: ";C$
2120 P$=C$
2130 GOTO 1940
2140 PRINT "NAME WAS: ";Q$;" NOW: ";C$
2150 Q$=C$
2160 GOTO 1940
2170 PRINT "OPTIONAL NAME WAS: ";R$;" NOW ";C$
2180 R$=C$
2190 GOTO 1940
2200 PRINT "ADDRESS WAS: ";S$;" NOW: ";C$
2210 S$=C$
2220 GOTO 1940
2230 PRINT "CITY, ETC WAS: ";T$;" NOW: ";C$
2240 T$=C$
2250 GOTO 1940
2260 PRINT "PHONE NO. WAS: ";U$;" NOW: ";C$
2270 U$=C$
2280 GOTO 1940
2290 PRINT "BIRTH MONTH WAS: ";V$;" NOW: "C$
2300 V$=C$
2310 GOTO 1940
2320 PRINT "BIRTHDAY WAS: ";W$;" NOW: ";C$
2330 W$=C$
```

```
2340  GOTO 1940
2350  IF U=1 THEN 2370
2360  GOTO 130
2370  X$=P$&Q$&R$&S$&T$&U$&V$&W$
2380  B=LEN(X$)
2390  IF B>128 THEN 2440
2400  PRINT #1,REC I:P$,Q$,R$,S$,T$,U$,V$,W$
2410  REM NOTICE MESSAGE
2420  PRINT "DISK RECORD UPDATED FOR ";Q$
2430  GOTO 130
2440  D=B-128
2450  PRINT "RECORD IS TOO LONG BY ";D;" CHARACTERS."
2460  PRINT "MORE REVISIONS ARE REQUIRED."
2470  GOTO 1940
2480  CLOSE #1
2490  GOTO 2510
2500  CLOSE #1:DELETE
2510  END
```

A PROGRAM TO COPY THE NAME & ADDRESS FILE ONTO ANOTHER DISK

```
 10 DIM A$(150),C$(150),D$(150),E$(150),F$(150),G$(150),H$(150),I$(150)
 20 GOSUB 550
 30 FOR I=0 TO 60
 40 GOSUB 580
 50 NEXT I
 60 CLOSE #1
 70 GOSUB 640
 80 INPUT B$
 90 GOSUB 550
100 FOR I=0 TO 60
110 GOSUB 610
120 NEXT I
130 CLOSE #1
140 GOSUB 670
150 INPUT B$
160 GOSUB 550
170 FOR I=61 TO 122
180 GOSUB 580
190 NEXT I
200 CLOSE #1
210 GOSUB 640
220 INPUT B$
230 GOSUB 550
240 FOR I=61 TO 122
250 GOSUB 610
260 NEXT I
270 CLOSE #1
280 GOSUB 670
290 INPUT B$
300 FOR I=0 TO 122
310 A$(I)=""
320 C$(I)=""
330 D$(I)=""
340 E$(I)=""
350 F$(I)=""
360 G$(I)=""
370 H$(I)=""
380 I$(I)=""
390 NEXT I
400 GOSUB 550
410 FOR I=123 TO 150
420 GOSUB 580
430 NEXT I
440 CLOSE #1
450 GOSUB 640
460 INPUT B$
470 GOSUB 550
480 FOR I=123 TO 150
490 GOSUB 610
```

```
500 NEXT I
510 CLOSE #1
520 PRINT "FILE COPY FUNCTION COMPLETED"
530 GOTO 690
540 REM FILE OPEN SUBROUTINE
550 OPEN #1:"DSK1.NAME&A",RELATIVE,INTERNAL,FIXED 136
560 RETURN
570 REM FILE READ SUBROUTINE
580 INPUT #1,REC I:A$(I),C$(I),D$(I),E$(I),F$(I),G$(I),H$(I),I$(I)
590 RETURN
600 REM FILE WRITE SUBROUTINE
610 PRINT #1,REC I:A$(I),C$(I),D$(I),E$(I),F$(I),G$(I),H$(I),I$(I)
620 RETURN
630 REM MESSAGE SUBROUTINE
640 PRINT "INSERT BACK-UP DISK.  PRESS ENTER WHEN READY."
650 RETURN
660 REM MESSAGE SUBROUTINE
670 PRINT "INSERT DISK WITH FILE TO BE COPIED.  PRESS ENTER WHEN READY."
680 RETURN
690 PRINT  "TO  PROCESS  THIS  FILE  COPY,  LOAD  AND  RUN  PROGRAM  NAMED
    NUPDATE."
700 PRINT  "INSERT  THIS  DISK  AFTER  PROGRAM  HAS  BEEN  LOADED  INTO  THE
    COMPUTER,"
710 PRINT "BUT BEFORE THE RUN INSTRUCTION."
720 END
```

The routine starting with line 300 and ending with 390 are to gain more memory space by replacing the file information in the numeric array with null characters - One null character per field. The array holds the file records in memory. The records that have been placed on the back-up file allready are replaced with null characters. That is done to prevent a program abort when available memory is exhausted. However, if you use the file and get a memory full abort message, the routine at lines 300 thru 390 will have to be repeated (a program revision), and inserted between lines 130 and 140, as follows:

```
NUM 131,1
131 FOR I=0 TO 60
132 A$(I)=""
133 C$(I)=""
134 D$(I)=""
135 E$(I)=""
136 F$(I)=""
137 G$(I)=""
138 H$(I)=""
139 I$(I)=""
140 NEXT I
141 GOSUB 670
```

Then you can resequence it if you want to. Save the program using the old name, COPYN&A before running the copy program over again.

SORT UTILITY - USER INSTRUCTIONS

In summary, your responsibilities will be to accomplish the following:

1. Enter the program into the computer from the program listing.
2. Save the program on a floppy disk.
3. Load the program into the computer, using TI Basic Mode, when ready to sort a file.
4. Fix the INPUT # statement to read your file, using the variable, JP(KQ) for the field to be sorted.
5. Run the program and be prepared to supply the name of the file.
6. Replace the disk containing the sort program with the disk containing the file to be sorted. The timing of this exchange is to be consistant with instructions that will appear on the monitor screen.
7. After the sort program reads your file, you are given the opportunity to change to a disk with enough space available to hold the sorted output file. Regardless of the size of your file, the sorted output file will occupy 57 sectors on a disk. If there is room on your file disk, that is where the output file should be stored, to make it possible for an update program to open both files in the same run without having to change disks.

This may seem strange, but it is true, this program will sort any length file under 501 records. At that point, the program quits reading the file. The program assumes the file contains the same record format for each record in the file. The program will have to be modified to read every second, or every third record in a file. That can be done easily and will be necessary, if the file you want sorted contains sets of records.

It is not necessary for you to tell the program the number of records in your file. Nor is it necessary for you to code the file OPEN statement. For fixed length files, the length of the file records is obtained from the file index. The only tailoring that must be done, is to be sure the field to be sorted is properly identified using the JP(KQ) variable.

On page 39 of my "Disk Memory System" TI Manual, field 2 of the Disk Index File record format is explained. Five file types are listed. The 1 and 5 file types will not be sorted by this program. Type 5 is for programs, and type 1 would be Sequential/Relative fixed length records, using display type data. The display data type classification would cause this program to abort after reading the last record in the file. The abort problem can be circumvented in TI Extended Basic, but that is not how this program was written. I doubt that many people would choose Display over Internal, when Internal is so easy to work with. File types 2, 3, and 4 may be sorted by this program, assuming the number of records do not exceed 501.

If the file you wish to sort is classified as File Type 2 (Display/Variable), it will be a sequential file. It will be necessary to revise line number 480 before running the sort utility program. If it is a file type 3, it will be necessary to revise line 630. Change line number 780 if the file type is a 4. These are the INPUT # statements which read your file. The changes required are to assign the variable codes so that the sort program reads the file. The program is only interested in reading the field in each record containing the number that is to be sorted. I'll explain the the type change you will have to make:

Assume the file records each contain 7 fileds. Starting at the left, the first field is numeric, the next numeric, the third alpha/numeric, the fourth alpha/numeric, the fifth, sixth, and seventh fields are numeric fields. The field to be sorted is the fifth field. The normal file read statement used in the file update program looks like this:

```
INPUT #1,REC I:A,B,A$,B$,C,D,E
```

It would look like that if the file is a random file. If the file is sequential, the normal file read statement looks like this:

```
INPUT #1:A,B,A$,B$,C,D,E
```

In either case, C is the field to use for the sort. The input statement in the Sort Utility program looks like this:

```
 INPUT #1,REC KQ:JP(KQ),A,B
Or:
     INPUT #1:JP(KQ),A,B
```

Neither instruction would read the file, because the third variable is for a numeric field, and the third field in the file is alpha/numeric. Also, JP(KQ) needs to be where C is located in the list of variables.

The revision required will be:

```
    INPUT #1,REC KQ:A,B,A$,B$,JP(KQ)
Or:
        INPUT #1:A,B,A$,B$,JP(KQ)
```

Note that there is no point in reading the information fields beyond the sort field. That is why JP(KQ) was not followed by two more variable codes.

When making the revisions, take care not to use variable codes which are being used by the Sort Utility program. The variable codes already assigned are as follows:

NUMERIC VARIABLES: CI DJ EK FL GM JP(KQ) KQ LR MS NT PV(KQ) QW RX TA UB

STRING VARIABLES: BY$ CZ$ DI$ EJ$

Though A and B appear in the INPUT # statements referred to, they are not used by the program, and you should feel free to use them in the revision. So, from the File Open statement in your maintenance program, compare file characteristics to the Disk Memory Manual to determine which program line number to revise. Load the program into TI Basic. type in the line number followed by the up arrow, to bring in the line to revise. Make the changes, then run the program. The change can be made permanent by saving the sort utility progaram before entering the run instruction. Otherwise, the change will be for the one run only.

THE SORT UTILITY OUTPUT FILE

The file produced by the utility program is 502 records long, and contains three numeric fields in each record. These are the key fields most likely to be used in random file update, or report processing programs. Record length is 27 characters, which makes it 3, 9 character numeric fields. Let's call them fields A,B,C. Field A in the SORTED file will contain a list of obsolete record numbers in your file. Random files are likely to have unused records, while sequential files probably will have none. Unused record numbers are detected by the sort program when the field value in the sort field is zero. Field A will contain the number 9999 when the field does not identify an unused record number.

Field B contains the number used in the sort taken from your file record sort field. To get assigned to this variable, the sort field number must be greater than zero. The number in this field will be zero from the end of your file, through record number 500.

Field C contains the record number in your file which corresponds to the number in field B. The zero record will be indicated by field B being greater than zero, even though field C contains a zero.

Record number 501 in the SORTED file contains file total information in three fields. Let's call them D,E,F. Field D is the total number of obsolete records in your file. Field E is the number of sorted records, and this relates to fields B and C. Field F is the total number of records in your file processed by the sort program. This should equal the sum of D plus E.

There is another program listing following the sort program, which I call SHOWSORT. This program is 46 lines long. It opens the sort utility output file and displays on the monitor excerpts from the file so that you should know exactly how the output file is structured. It also provides an example of the instructions you will need to use to read the file. If your concept of the structure of the SORTED file is still a little hazy, enter and run this program for a clear understanding of the sorted file.

SUGGESTED USES FOR THE SORT UTILITY

This sort program is intended primarily for use with random files that are defined using the Internal format. The sort program will sort the key field for sequential files with variable length records, but I have been unable to figgure out a way you could use the output file. If those are the types of files you are concerned about, it will be necessary to adapt the sort routine to your particular file. One suggestion - You could convert your file to a sequential or random fixed length, internal file. Then process that output file as a random file for reports, etc.. It wouldn't take much of a program to make that conversion, and then by working with it as a relative file, there would be no problem producing reports in the proper sequence. To make that clearer, let me give you a fictitious example:

Say I have a sequential file containing three fields: A,B,C$, and C$ could be any length. I want the sort to be on field B. I would run the sort program, changing line 480 to read: INPUT #1:A,JP(KQ). This would produce an output SORTED file. To convert my file to a random, fixed length file, I would write the following program:

```
10    OPEN #1:"DSK1.SORTED",RELATIVE,INTERNAL,FIXED 27
20    INPUT #1,REC 501:A,B,C
30    CLOSE #1
20    OPEN #1:"DSK1.MYFILE",DISPLAY,VARIABLE
30    C=B-1
40    OPEN #2:"DSK1.MYFIEE",RELATIVE,INTERNAL,FIXED 254
50    FOR I=0 TO C
60    INPUT #1:A,B,C$
70    PRINT #2,REC I:A,B,C$
80    NEXT I
90    CLOSE #1
100   CLOSE #2
110   END
```

You get the idea, anyway. Additional instructions may be required, if disk space is a problem. If necessary, you could read from file #1, create an array, then change disks and write the array onto file #2. Then repeat the process, till file #1 has been completely transferred. The example program is not going to handle all situations, but is an indicator of what can be done to adapt to the sort utility. A good programmer never gives up till all possibilities have been exhausted.

Now that all of your files have been converted to Type 3 files, let's go on from there. The information from SORTED can be read into your update or report program as an array, or you could read the file sequentially. Let's say the only thing you are interested in is record number in your file. The sorted record numbers. Let's use the previous three field file example. This is a program to print the complete file in sequence by field B.

```
10    DIM D(500)
20    OPEN #1:"DSK1.SORTED",RELATIVE,INTERNAL,FIXED 27
30    INPUT #1,REC 501:A,B,C
40    C=B-1
50    FOR I=0 TO C
```

```
60    INPUT #1,REC I:A,B,D(I)
70    NEXT I
80    CLOSE #1
90    INPUT "YOUR CHANCE TO INSERT THE FILE DISK: ":A$
100   OPEN #1:"DSK1.MYFIEE",RELATIVE,INTERNAL,FIXED 254
110   FOR I=0 TO C
120   INPUT #1,REC D(I):A,B,C$
130   PRINT A;B;C$
140   NEXT I
150   CLOSE #1
160   END
```

I'm sure your print program will be more detailed than that, but this illustrates how you can use the SORTED file information as an array in your print program. It is not necessary to use all fields of information carried in the SORTED file.

For what it is worth, here is my own opinion. Any sort program will be a time consuming program run. If you have a relative file, you could run the sort program to establish the initial sort sequence, then code in routines to maintain the sort seqence from that point on. As new records are added to the file, adjust the arrays to reflect the change. As records are deleted, again adjust the array. At the end of a processing run, save the arrays and change the total record, 501. At the beginning of the program run, read in the SORTED file as an array. By maintaining the arrays so they are always current, you can avoid having to run the sort utility program to print a report. I'll admit those are brain buster type programming routines to code, but once developed, you will find the arrays can be used in many way, besides printing reports in sequence. If you try that type of programming, and give up, I have developed such a routine in the SCHEDULE program, which I sell as Item No. 5, for $30.00, delivered.

SORT UTILITY PROGRAM

FOR FILES CREATED IN TI BASIC AND UNDER 501 RECORDS

```
 10  PRINT "SORT UTILITY PROGRAM":" "
 20  PRINT "COPYRIGHT 1982  J. H. HARVEY 159 DOVER RD., SPARTANBURG,
      SOUTH CAROLINA, 29301":" "
 30  DIM JP(500),PV(500)
 40  PRINT "INSERT DISK CONTAINING THE FILE TO BE SORTED.":" "
 50  INPUT "PRESS THE ENTER KEY WHEN READY: ":BY$
 60  CALL CLEAR
 70  INPUT "NEED THE NAME OF THE FILE TO BE SORTED: ":CZ$
 80  PRINT
 90  OPEN #1:"DSK1.",INPUT,RELATIVE,INTERNAL
100  FOR CI=0 TO 127
110  IF DJ=1 THEN 170
120  INPUT #1,REC CI:DI$,EK,FL,GM
130  IF DI$="" THEN 160
140  IF DI$=CZ$ THEN 160
150  GOTO 170
160  DJ=1
170  NEXT CI
180  IF DI$=CZ$ THEN 260
190  PRINT "FILE NAME COULD NOT BE FOUND. "; CZ$:" "
200  INPUT "PRESS ENTER TO TRY AGAIN, OR ENTER A 9 TO STOP PROCESSING:
      ":BY$
210  PRINT
220  IF BY$="9" THEN 1350
230  CLOSE #1
240  DJ=0
250  GOTO 40
260  EJ$="DSK1."&CZ$
270  CLOSE #1
280  IF EK=1 THEN 330
290  IF EK=2 THEN 430
300  IF EK=3 THEN 580
310  IF EK=4 THEN 730
320  IF EK=5 THEN 370
330  PRINT "THIS IS THE TYPE FILE MY T I BASIC SORT ROUTINE CANNOT
      HANDLE.":" "
340  PRINT "HARVEY'S EXTENDED BASIC VERSION OF THE SORT UTILITY IS THE
      VERSION YOU NEED.":" "
350  PRINT "TERMINATING THE PROGRAM.":" "
360  GOTO 1440
370  PRINT "THE NAME ";CZ$;" BELONGS TO A PROGRAM. TO GET THE RIGHT
      NAME, CHECK THE INDEX, OR THE FILE":" "
380  INPUT "OPEN STATEMENT. PRESS ENTER TO TRY AGAIN, OR ENTER A 9 TO
      STOP PROCESSING: ":BY$
390  PRINT
400  IF BY$="9" THEN 1440
410  DJ=0
```

```
420  GOTO 40
430  TA=480
440  GOSUB 1370
450  OPEN #1:EJ$,DISPLAY,VARIABLE
460  FOR KQ=0 TO 500
470  IF LR=1 THEN 530
480  INPUT #1:JP(KQ),A,B
490  MS=MS+1
500  IF EOF(1)<>0 THEN 520
510  GOTO 530
520  LR=1
530  NEXT KQ
540  CLOSE #1
550  LR=0
560  GOSUB 1390
570  GOTO 870
580  TA=630
590  GOSUB 1370
600  OPEN #1:EJ$,RELATIVE,INTERNAL,FIXED GM
610  FOR KQ=0 TO 500
620  IF LR=1 THEN 680
630  INPUT #1,REC KQ:JP(KQ),A,B
640  MS=MS+1
650  IF EOF(1)<>0 THEN 670
660  GOTO 680
670  LR=1
680  NEXT KQ
690  CLOSE #1
700  LR=0
710  GOSUB 1390
720  GOTO 870
730  TA=780
740  GOSUB 1370
750  OPEN #1:EJ$,INTERNAL,VARIABLE
760  FOR KQ=0 TO 500
770  IF LR=1 THEN 830
780  INPUT #1:JP(KQ),A,B
790  MS=MS+1
800  IF EOF(1)<>0 THEN 820
810  GOTO 830
820  LR=1
830  NEXT KQ
840  CLOSE #1
850  LR=0
860  GOSUB 1390
870  FOR KQ=0 TO MS-1
880  IF JP(KQ)=0 THEN 900
890  GOTO 920
900  PV(NT)=KQ
910  NT=NT+1
920  NEXT KQ
930  REM
```

```
940  PRINT "SORTING - THIS WILL TAKE 5 TO 90 MINUTES TO COMPLETE":" "
950  PRINT "THE DISK DRIVE LIGHT WILL GO ON AND OFF TILL THE SORT IS
      OVER. PLEASE WAIT.":" "
960  OPEN #1:"DSK1.SORTED",RELATIVE,INTERNAL,FIXED 27
970  FOR KQ=NT TO 500
980  PV(KQ)=9999
990  NEXT KQ
1000 IF NT=MS THEN 1090
1010 FOR KQ=0 TO MS-1
1020 IF JP(KQ)=0 THEN 1070
1030 IF QW=0 THEN 1050
1040 IF QW<=JP(KQ) THEN 1070
1050 QW=JP(KQ)
1060 RX=KQ
1070 NEXT KQ
1080 IF QW=0 THEN 1160
1090 PRINT #1,REC UB:PV(UB),QW,RX
1100 UB=UB+1
1110 JP(RX)=0
1120 QW=0
1130 RX=0
1140 IF UB=MS THEN 1160
1150 GOTO 1000
1160 IF UB=500 THEN 1250
1170 QW=0
1180 RX=0
1190 FOR KQ=UB TO 500
1200 PRINT #1,REC KQ:PV(KQ),QW,RX
1210 NEX KQ
1220 IF NT=MS THEN 1240
1230 GOTO 1250
1240 UB=0
1250 PRINT #1,REC 501:NT,UB,MS
1260 CLOSE #1
1270 PRINT
1280 PRINT "ACTIVITY SUMMARY":" "
1290 PRINT "OBSOLETE RECORDS ARE OF INTEREST WITH RANDOM FILE
      PROCESSING.":" "
1300 PRINT "NO. OBSOLETE RECORDS IN THE FILE: ";NT:" "
1310 PRINT "NO. ACTIVE RECORDS SORTED: ";UB:" "
1320 PRINT "NUMBER OF RECORDS IN YOUR FILE: ";MS:" "
1330 PRINT "HAVE A GOOD DAY!":" "
1340 GOTO 1440
1350 CLOSE #1
1360 GOTO 1440
1370 PRINT " ":"PROCESSING ON THE ASSUMPTION THAT YOU HAVE REVISED LINE
      NO.: ";TA;" TO READ YOUR FILE.":" "
1380 RETURN
1390 PRINT "YOUR FILE HAS BEEN READ.":" "
1400 PRINT "THE NEXT USE OF THE DISK DRIVE WILL BE TO RECORD THE SORT
      FILE.":" "
1410 INPUT "YOUR CHANCE TO CHANGE DISKS. PRESS THE ENTER KEY WHEN READY:
      ":BY$
1420 PRINT
1430 RETURN
1440 END
```

THE SHOWSORT PROGRAM

If you have problems visualizing the structure and format of the file produced by the sort utility program, the following short program will display the pertinent 3 parts of the file, which should give you a clear understanding of how the file is constructed. It also provides an example of the code required to read the file.

```
10    CALL CLEAR
20    PRINT "PROGRAM DISPLAYS THE SORTED FILE RECORDS ON THE MONITOR.":"
       "
30    OPEN #1:"DSK1.SORTED",RELATIVE,INTERNAL,FIXED 27
40    INPUT #1,REC 501:A,B,C
50    PRINT "NO. OBSOLETE RECORDS";A:" "
60    PRINT "NO. SORTED RECORDS: ";B:" "
70    PRINT "NO. RECORDS IN YOUR FILE: ";C:" "
80    PRINT "COLUMN 1 = SORT FILE RCD. NO.":" "
90    PRINT "COLUMN 2 = OBSOLETE RCD. NO. IN YOUR FILE.":"WILL BE 9999 IF
       NOT A RCD. NUMBER.":" "
100   PRINT "COLUMN 3 = A NO. FROM YOUR FILE RCD. SORT FIELD.":" "
110   PRINT "COLUMN 4 = THE RECORD NO. IN YOUR FILE FOR THE NO. SHOWN IN
       COLUMN 3.":" "
120   INPUT "PRESS ENTER KEY WHEN READY TO CONTINUE: ":A$
130   CALL CLEAR
140   FOR I=0 TO 22
150   INPUT #1,REC I:D,E,F
160   PRINT I;D;E;F
170   NEXT I
180   INPUT "PRESS ENTER TO CONTINUE: ":A$
190   IF A>22 THEN 210
200   GOTO 320
210   G=A-5
220   H=G+22
230   IF H>501 THEN 400
240   CALL CLEAR
250   PRINT "SHOWING END OF OBSOLETE RCDS":" "
260   FOR I=G TO H
270   INPUT #1,REC I:D,E,F
280   PRINT I;D;E;F
290   NEXT I
300   INPUT "PRESS ENTER TO CONTINUE: ":A$
310   CALL CLEAR
320   IF J=1 THEN 400
330   IF J=2 THEN 450
340   J=1
350   G=B-5
```

```
360  H=G+22
370  IF H>500 THEN 400
380  PRINT "SHOWING THE END OF FILE FOR YOUR RECORDS: ":" "
390  GOTO 260
400  J=2
410  G=490
420  H=501
430  PRINT "SHOWING FILE TOTAL RCD. 501. COLUMNS 2, 3, & 4 ARE YOUR FILE
      TOTALS. ":" "
440  GOTO 260
450  CLOSE #1
460  END
```

MODIFICATION TO THE TI BASIC SORT UTILITY PROGRAM
CONVERSION TO TI EXTENDED BASIC

BY JAMES HARVEY, 159 DOVER RD., SPARTANBURG, S.C., 29301

The following type files will be passed over by the TI Basic sort utility as one which will not be sorted. The file OPEN statement would define those files as sequential or relative, using the display data type, and fixed length records. The program would abort if it attempted to sort that type of file. The ON ERROR instruction can be used to circumvent the problem, but the instruction is only accepted when in the TI Extended Basic mode. I can guess at the cause of the abort. When the sort program opens type 1 files as a relative file, the EOF function does not work, (it was not intended to be used with relative files) with the result that the program attempts to read beyond the end of the file. That is a no, no, and triggers an abort of the sort program.

In TI Extended Basic, the same problem exists, but the abort condition can be used as a way to indicate when the end of the file has been reached. The ON ERROR instructions allow processing to continue, and prevents the run abort, as long as the file is not closed. The following modification to the TI Basic Sort program includes the ON ERROR instruction and does not close your file. That means you must close the file manually when the sort program is finished. This may be accomplished either by running another program, or entering BYE. Do not turn off the computer equipment till one or the other of those commands are executed.

This modification will work if the following two conditions are true:

1. You have the TI Basic Sort program stored on a floppy disk.
2. You used the same program line numbers as are on the printed program listing.

ENTERING THE MODIFICATION

STEP 1 From a cold start, turn on the equipment and get into TI Extended Basic.

STEP 2 Load the TI Basic version of the sort program into the computer from disk storage.

STEP 3 Delete program line numbers 330, 340, 350 and 360. That is accomplished by typing in the line numbers and pressing the Enter key.

STEP 4 Type in the following command, then press the Enter key: NUM 330,1

STEP 5 Enter the following program instructions:

```
330   TA=336
331   GOSUB 1370
332   OPEN #2:EJ$,RELATIVE,DISPLAY,FIXED GM
333   ON ERROR 345
334   FOR KQ=0 TO 500
335   IF LR=1 THEN 341
336   INPUT #2,REC KQ:JP(KQ),A,B
337   MS=MS+1
338   IF EOF(2)<>0 THEN 340
339   GOTO 341
340   LR=1
341   NEXT KQ
```

```
342   LR=0
343   GOSUB 1390
344   GOTO 870
345   LR=1
346   ON ERROR 334
347   RETURN
```

STEP 6 Resequence the program, if you wish, but all the TA variable values will have to be changed to specify the revised line numbers. That would be a good reason not to resequence the program.

STEP 7 Save the program on disk, using another program name. You want to finish with two versions of the program stored on a disk, because with this modification, the program will not run in TI Basic.

RECORD NUMBER CROSS-REFERENCE FORM

RCD.	LAST NAME	LAST NAME	LAST NAME
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			

RECORD NUMBER CROSS-REFERENCE FORM

RCD.	LAST NAME	LAST NAME	LAST NAME
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			
38			
39			
40			
41			
42			
43			
44			
45			
46			
47			
48			
49			
50			
51			
52			
53			

RECORD NUMBER CROSS-REFERENCE FORM

RCD.	LAST NAME	LAST NAME	LAST NAME
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			
69			
70			
71			
72			
73			
74			
75			
76			
77			
78			
79			
80			

RECORD NUMBER CROSS-REFERENCE FORM

RCD.	LAST NAME	LAST NAME	LAST NAME
81			
82			
83			
84			
85			
86			
87			
88			
89			
90			
91			
92			
93			
94			
95			
96			
97			
98			
99			
100			
101			
102			
103			
104			
105			
106			
107			

RECORD NUMBER CROSS-REFERENCE FORM

RCD.	LAST NAME	LAST NAME	LAST NAME
108			
109			
110			
111			
112			
113			
114			
115			
116			
117			
118			
119			
120			
121			
122			
123			
124			
125			
126			
127			
128			
129			
130			
131			
132			
133			
134			

RECORD NUMBER CROSS-REFERENCE FORM

RCD.	LAST NAME	LAST NAME	LAST NAME
135			
136			
137			
138			
139			
140			
141			
142			
143			
144			
145			
146			
147			
148			
149			
150			