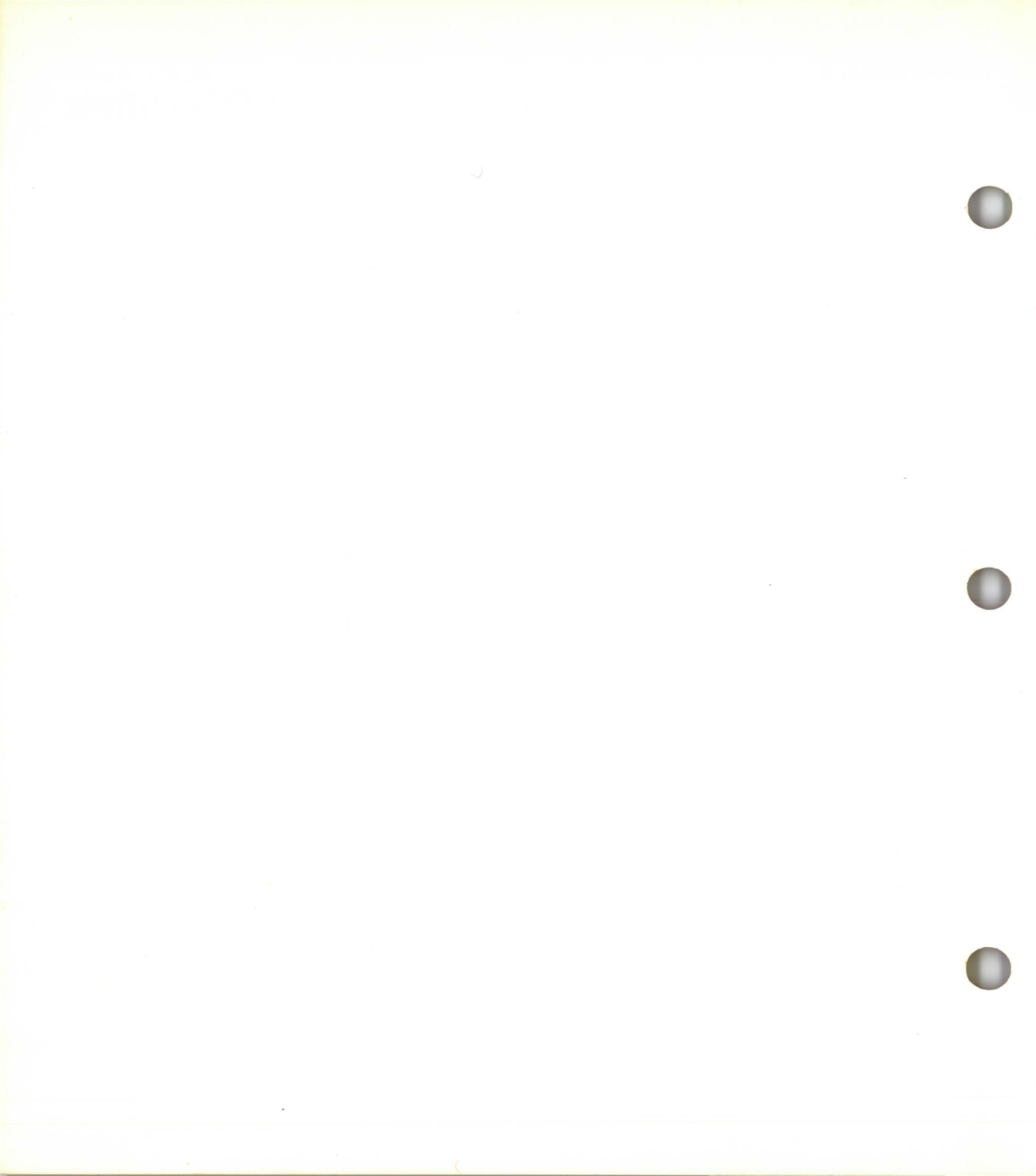


VI. Data Files and Diskettes

IBM

Learning System/23 BASIC



VI. Data Files and Diskettes



Learning System/23 BASIC

Second Edition (August 1981)

This is a major revision of, and obsoletes SA34-0126-0. Because the changes and additions are extensive, this manual should be reviewed in its entirety.

Use this publication only for the purpose stated in the Preface.

Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below. Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Information Development, Department 27T, P.O. Box 1328, Boca Raton, Florida 33432. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

VI. Data files and diskettes

Contents

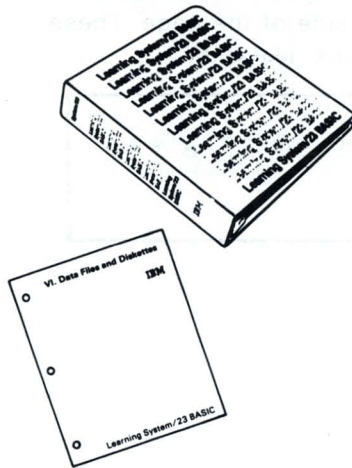
About this book	v
Chapter 1. Data files—DISPLAY	1-1
Introduction.....	1-1
Creating a file at the keyboard.....	1-2
File status and the DIR command.....	1-5
Correcting an error.....	1-6
Creating a file in a program	1-7
Assigning values from a file.....	1-12
Chapter summary.....	1-14
Exercises	1-15
Answers	1-17
Chapter 2. INTERNAL files—sequential access	2-1
Introduction.....	2-1
Creating a file.....	2-2
Assigning values from a file.....	2-11
Chapter summary.....	2-15
Exercises	2-16
Answers	2-19
Chapter 3. INTERNAL files—relative access	3-1
Introduction.....	3-1
The OPEN statement	3-2
The WRITE statement.....	3-4
Assigning values from a file.....	3-7
Chapter summary.....	3-10
Exercises	3-11
Answers	3-13
Chapter 4. Repositioning a file	4-1
Introduction.....	4-1
The RESTORE statement	4-3
The REREAD statement	4-13
The REWRITE statement	4-15
The DELETE statement	4-16

VI. Data files and diskettes

Contents (continued)

The CLOSE statement.....	4-17
Controlling file access errors.....	4-18
Chapter summary.....	4-25
Exercises.....	4-26
Answers.....	4-29
Chapter 5. INTERNAL files—key-indexed access	5-1
Introduction.....	5-1
What is a key-indexed file?.....	5-2
Setting up a key-indexed file.....	5-4
Creating an index file.....	5-6
Reading a record from a key-indexed file.....	5-15
Writing and deleting records in a key-indexed file.....	5-17
Chapter summary.....	5-21
Exercises.....	5-22
Answers.....	5-24
Chapter 6. Diskettes and diskette drives	6-1
Introduction.....	6-1
The share state.....	6-2
Device sharing information.....	6-4
File sharing information.....	6-6
The DIR and PROTECT commands.....	6-8
Other BASIC commands.....	6-10
Chapter summary.....	6-11
Exercises.....	6-12
Answers.....	6-13

About this book



In Books I through V of this course, you learned the fundamental commands and statements required to program your System/23. You now know how to enter and change a program. You also know how to run a program and store it on a diskette.

In your programs you can:

- Display or print information
- Perform arithmetic operations
- Branch to specified program sections
- Assign values to variables and arrays
- Use subroutines, functions, and loops

In Book VI, you will learn about one more helpful programming feature: the *data file*. A data file is used to store information on a diskette.

You will be learning about diskettes and saving copies of files. Therefore, make sure you still have your diskette inserted in diskette drive 1 (or drive 3, if you are using a 5322-0) before starting this book.

One more note: So far in this course, whenever we asked you to enter a command or statement, it was printed in green in your book. Beginning in Book VI, we will often ask you to enter several long statements.

VI. Data files and diskettes

About this book (continued)

When we ask you to enter any long statements, we will still print the statements in this book in green. However, we will start the lines on the far lefthand side of the page. These statements will be enclosed in a box, like this:

```
CLEAR  
10 PRINT "BOOK VI IN THE COURSE LEARNING SYSTEM/23 BASIC"  
20 END  
RUN
```


Chapter 1. Data files - DISPLAY

Introduction

In this chapter, you will learn about *data files*. A *file* is a collection of related items that are stored together. In Book I of this course, you learned that a *program file* is a collection of program statements. *Data* is known information. A *data file* is a collection of related data items.

Objectives

Upon completion of this chapter, you should be able to do the following:

- Assign a name to a data file.
- Enter a data file into the work area after first using the CLEAR DATA command.
- Save a copy of a file by using the SAVE command.
- Load a display file into the work area by using the LOAD DATA command.
- List the contents of a file by using the LIST command.
- Correct an error in a program that uses files by using the DIR, PROTECT, and FREE commands.
- Open a display file by using the OPEN statement.
- Copy data into a file by using the PRINT statement.
- Input data from a file by using the LINPUT statement.

If you are familiar with these tasks, try the exercises at the end of this chapter. If not, read through the chapter before going on to the exercises.

Data files - DISPLAY

Creating a file at the keyboard

In Book II, you learned how to use READ and DATA statements to assign values to variables. The data values were an actual part of the program.

```
10 DATA 5,10,15
20 READ A,B,C
30 PRINT A;B;C
40 END
```

You can also store data values in a file, instead of making the values an actual part of a program. As the data values in a file are needed, they are assigned to variables in a program. The data in a file can also be used by more than one program.

Let's look at an example. Enter the following:

```
CLEAR DATA
```

As you know, the CLEAR command clears the work area of all statements previously entered. The word *DATA* tells your System/23 that you are going to enter *data*, not a program.

Now, we're going to enter some names and addresses, which we will save in a data file. Enter the AUTO command, and see what happens:

```
AUTO
00010: _
```

```
AUTO
```

As you can see, the line number 00010 is followed by a colon. All of the information in a data file is entered with line numbers like this (00010:, 00020:, etc.).

Now, on lines 10, 20, 30, and 40, enter the following:

```
00010:GENERAL BUSINESS
00020:WHITE PLAINS, NY
00030:GENERAL SYSTEMS
00040:ATLANTA, GA
00050:_
```

```
00010:GENERAL BUSINESS
00020:WHITE PLAINS, NY
00030:GENERAL SYSTEMS
00040:ATLANTA, GA
```

To stop the AUTO operation, press the Scroll Up key. Remember that this is also how you stop the AUTO operation when you are entering a program.

Now, you have a data file in your work area. Enter the LIST command:

```
00010:GENERAL BUSINESS
00020:WHITE PLAINS, NY
00030:GENERAL SYSTEMS
00040:ATLANTA, GA
-
```

```
LIST
```

As you can see, this data file is listed just like a program, except that the listing of your data file has colons after the line numbers.

You can change a data file in the work area the same way you change a BASIC program. For example,

- To delete line 30, you enter DEL 30.
- To add another line of data, you enter the line number (followed by a colon) and the data.

Let's try to add another line of data. Enter the following:

```
00050:OFFICE PRODUCTS
```

Now, list your file:

```
00010:GENERAL BUSINESS
00020:WHITE PLAINS, NY
00030:GENERAL SYSTEMS
00040:ATLANTA, GA
00050:OFFICE PRODUCTS
-
```

```
LIST
```

Data files - DISPLAY

Creating a file at the keyboard (continued)

Your turn!

Enter a command to remove line 50 from your file.

Answer: _____

You should have entered:

DEL 50

List the file in the work area again to be sure that line 50 was deleted:

LIST

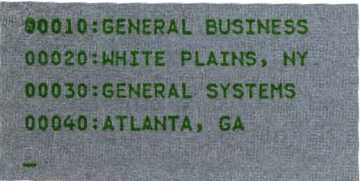
Now, make sure that your diskette is inserted in the diskette drive, and enter:

SAVE NAME.ADDRESS//1 (if your
diskette is in drive slot 1)

or

SAVE NAME.ADDRESS//3 (if your
diskette is in drive slot 3)

NAME.ADDRESS is the name that we chose for this data file. Remember from Book I that a filename can be from 1 to 17 characters long, including periods. Names longer than eight characters must consist of two or more simple filenames separated by periods. A simple filename is from 1 to 8 characters long.



```
00010:GENERAL BUSINESS
00020:WHITE PLAINS, NY
00030:GENERAL SYSTEMS
00040:ATLANTA, GA
-
```

File status and the DIR command

On the following pages, we are going to show you how to use data files in a BASIC program. Before we go on, however, you should know something about the status of your files.

In *Learning to Use System/23*, you learned how to use the DIR command. Enter the DIR command now, to see what files are stored on your diskette:

If your diskette is in drive slot 1, Type: DIR 1.
If your diskette is in drive slot 3, Type: DIR 3.

```
DIR 1
VOLID 1 0269312 0046 0000 512
09 SALESTAX      0008192 0004096 0001
09 NAME         0008192 0004096 0001
09 PRICE        0008192 0004096 0001
05 NAME.ADDRESS 0000512 0000512 0001
-
```

If the letters NS, ISI, OSI, OSH, or ISH appear in the status indicator (at the end of each file's information), then that file has been left open. The word *OPEN* will be discussed later.

For example, if the file NAME.ADDRESS had somehow been left open, it might look like this in the DIR:

```
05 NAME.ADDRESS      0000512 0000512 0001  NS
```

Before you could use the file NAME.ADDRESS again, you would have to close it. (The word *CLOSE* will be discussed later.)

Data files - DISPLAY

Correcting an error

If you get an error while running a program that uses a file, you should do the following:

1. Enter GO END to end the program.
2. Use the DIR command to see if the file exists and if it is open (ISI, OSI, NS, OSH, or ISH in the DIR).
3. If you are running a program that creates a new file, enter FREE filename. This command removes the file from your diskette. Then skip to step 5.
4. If the file is not open, skip to step 5. If the file is open, enter PROTECT filename,RELEASE. This command closes the file and lets you access the file again in your program.
5. Make any necessary corrections to your BASIC program.
6. Rerun the program.

You should already be familiar with the PROTECT and FREE commands. For more information, see Chapter 6, "Commands," in your *Operator Reference*.

Note: If a file is shared with another system, be sure it is not in use. If you close a file using PROTECT when a file is being used, you may cause the file to be lost.

Now, if you should run into problems in the examples that follow, you will know what to do. Don't forget, too, that you can always refer to your *Messages* manual when you get an error.

Creating a file in a program

You have seen how to enter a data file from the keyboard. This data file can now be used for different programs. Now we're going to show you how to create a data file while you are running a program. All set? Then enter:

```
CLEAR
10 DIM NAME$*24,ADDRESS$*40
20 OPEN #1:"NAME=ADDRESS2//1,SIZE=0",DISPLAY,OUTPUT
30 FOR I=1 TO 2
40 PRINT "ENTER NAME"
50 INPUT NAME$
60 PRINT "ENTER ADDRESS"
70 INPUT ADDRESS$
80 PRINT #1:NAME$
90 PRINT #1:ADDRESS$
100 NEXT I
110 END
```

Before we run this program, let's look at a few things. First of all, look at line 10.

```
10 DIM NAME$*24,ADDRESS$*40
```

Remember from Book IV that this means you can enter up to 24 characters for the name and up to 40 characters for the address.

Now let's look at line 20.

```
20 OPEN #1:"NAME=ADDRESS2//1,SIZE=0",DISPLAY,OUTPUT
or
20 OPEN #1:"NAME=ADDRESS2//3,SIZE=0",DISPLAY,OUTPUT
```

This statement creates a file called ADDRESS2 on the diskette in diskette drive 1 or 3. The OPEN statement makes a file available to a BASIC program.

Data files - DISPLAY

Creating a file in a program (continued)

```
20 OPEN #1:"NAME=ADDRESS2//1,SIZE=0",DISPLAY,OUTPUT
```

The #1 is a *file reference* number. It identifies a file throughout a program. You can see that we use this same file reference number in lines 80 and 90:

```
80 PRINT #1:NAME$  
90 PRINT #1:ADDRESS$
```

A file reference number can be any number from 1 through 127. It *must* be preceded by a pound sign (#).

Each program can access many files. If you were using three files in a program, you could refer to the files as #1, #2, and #127. The choice of numbers is unimportant as long as each file has its own unique file reference number.

```
20 OPEN #1:"NAME=ADDRESS2//1,SIZE=0",DISPLAY,OUTPUT
```

Everything between the quotation marks is the *file-id*. Note that the colon and the quotation marks are required.

The file-id identifies a file on a diskette. The name of the file is ADDRESS2. Note that the NAME= is required.

The //1 (or //3) indicates that you are opening a file on diskette drive 1 (or drive 3). Your System/23 uses this number to know where to put the file.

20 OPEN #1:"NAME=ADDRESS2//1,SIZE=0",DISPLAY,OUTPUT

The SIZE= reserves space on the diskette to hold your data. It also indicates that you are creating a new file. At this point, don't worry about what value to enter. Just enter SIZE=0. After you become more familiar with your System/23, you can refer to "OPEN statement" in your *BASIC Language Reference* manual for more information.

20 OPEN #1:"NAME=ADDRESS2//1,SIZE=0",DISPLAY,OUTPUT

The word DISPLAY indicates the type of file that you are using. A DISPLAY file is one that you can load into the work area and list on the screen.

The word OUTPUT indicates that you are going to copy data into the file from your program. You copy data into the file with PRINT statements.

```
80 PRINT #1:NAME$
90 PRINT #1:ADDRESS$
```

These statements "print" or copy data into your file, just like the statement PRINT N\$ displays data on the screen.

Data files - DISPLAY

Creating a file in a program (continued)

List your program:

```
00010 DIM NAME$*20, ADDRESS$*40
00020 OPEN #1:"NAME=ADDRESS2//1,SIZE=0",DISPLAY,OUTPUT
00030 FOR I=1 TO 2
00040 PRINT "ENTER NAME"
00050 INPUT NAME$
00060 PRINT "ENTER ADDRESS"
00070 INPUT ADDRESS$
00080 PRINT #1:NAME$
00090 PRINT #1:ADDRESS$
00100 NEXT I
00110 END
```

LIST

Now run the program and enter these responses:

RUN

ENTER NAME

?GENERAL BUSINESS
ENTER ADDRESS

? "WHITE PLAINS, NY"
ENTER NAME

?GENERAL SYSTEMS
ENTER ADDRESS

? "ATLANTA, GA"

Notice that we need to use quotation marks around "WHITE PLAINS, NY" and "ATLANTA, GA" because of the commas.

If you had a problem with this example, go back and follow the instructions on page 1-6.

```
RUN
ENTER NAME

?GENERAL BUSINESS
ENTER ADDRESS

?"WHITE PLAINS, NY"
ENTER NAME

?GENERAL SYSTEMS
ENTER ADDRESS

?"ATLANTA, GA"
```

You now have two files on your diskette with the same data:

- NAME.ADDRESS—created at the keyboard
- ADDRESS2—created in a program

Just to be sure that these two files are the same, let's load them into the work area. Enter the following:

```
LOAD NAME.ADDRESS,DATA
00010:GENERAL BUSINESS
00020:WHITE PLAINS, NY
00030:GENERAL SYSTEMS
00040:ATLANTA, GA
-
```

LOAD NAME.ADDRESS,DATA

The word DATA tells your System/23 that you are loading a *data* file, not a program. If you don't enter the word DATA, your System/23 thinks you are trying to load a program file.

Notice that the contents of the file are listed on the screen when you load the file into the work area.

Your turn!

Enter a command to load the second data file into the work area.

Answer: _____

You should have entered:

```
LOAD ADDRESS2,DATA
00010:GENERAL BUSINESS
00020:WHITE PLAINS, NY
00030:GENERAL SYSTEMS
00040:ATLANTA, GA
-
```

LOAD ADDRESS2,DATA

As you can see, the two files look the same.

Data files - DISPLAY

Assigning values from a file

Now we're going to write a program that uses the data you saved in the ADDRESS2 file. Enter the following:

```
CLEAR
10 DIM N$*24,A$*40
20 OPEN #1:"NAME=ADDRESS2",DISPLAY,INPUT
30 FOR I=1 TO 2
40 LINPUT #1:N$
50 LINPUT #1:A$
60 PRINT N$,A$
70 NEXT I
80 END
```

Look at the statement in line 20.

```
20 OPEN #1:"NAME=ADDRESS2",DISPLAY,INPUT
```

This OPEN statement is a little different from the statement you used to create the file. First of all, you must not include a SIZE= when you open a file that already exists on your diskette.

Also, you don't need the //1 or //3. Your System/23 finds the file since it has already been created.

In this statement, the word INPUT indicates that you are going to input data from the file into your program. Notice that we inputted data from a display file into a program with LINPUT statements.

```
40 LINPUT #1:N$
50 LINPUT #1:A$
```

The LINPUT statement inputs an entire string, including any commas, and assigns that value to a character variable. You are inputting values for the variables N\$ and A\$. Notice that you do not have to use the same variables that you used in the program that created the file.

Go ahead and run the program:

```

RUN
GENERAL BUSINESS      WHITE PLAINS, NY
GENERAL SYSTEMS      ATLANTA, GA

```

RUN

You can see that the data values are displayed.

Now, save your program in a file called ADDRESS.PROGRAM. Do you remember how to do this? Enter:

```

SAVE ADDRESS.PROGRAM//1
OR
SAVE ADDRESS.PROGRAM//3

```

Note: This program will also work with the data file NAME.ADDRESS that you entered at the keyboard. Let's try it.

Enter this change. (Remember that you can list a program and make a change to an existing line.)

```

20 OPEN #1:"NAME=NAME.ADDRESS",DISPLAY,INPUT

```

And run your program:

```

RUN
GENERAL BUSINESS      WHITE PLAINS, NY
GENERAL SYSTEMS      ATLANTA, GA

```

RUN

Notice that the results are the same.

Data files - DISPLAY

Chapter summary

You can store data in a file in order to use that data in one or more programs. The data in a display file can be loaded into the work area and listed on the screen.

You can create a display file in one of two ways:

- At the keyboard by using CLEAR DATA and SAVE
- In a program by using OPEN DISPLAY,OUTPUT and PRINT

You can retrieve data from a display file in one of two ways:

- At the keyboard by using LOAD DATA and LIST
- In a program by using OPEN DISPLAY,INPUT and LINPUT

A filename is from 1 to 17 characters long, including periods. If a filename is longer than eight characters, it must be divided into two or more simple filenames separated by periods. Simple filenames are from 1 to 8 characters long.

Exercises

Question 1

Which of the following are valid filenames?

- a. SAVINGS
- b. NAMES.AND.ADDRESSES
- c. NAME.10
- d. PROGRAM.A1
- e. 2.PROGRAM

Question 2

Match the following commands with the function they perform.

- | | |
|------------------|---|
| _____ CLEAR DATA | a. Display the contents of the work area. |
| _____ SAVE | b. Store a copy of a file onto a diskette. |
| _____ LIST | c. Clear the work area and allow data to be entered. |
| _____ LOAD DATA | d. Place a copy of a data file back into the work area. |

Data files - DISPLAY

Exercises (continued)

Question 3

Add a statement to the following program, on line 30, that will place the data 1000 MAIN STREET into the file STREET.

```
10 OPEN #1:"NAME=STREET//1,SIZE=0",DISPLAY,OUTPUT
20 S$="1000 MAIN STREET"
40 END
```

Answer: _____

Question 4

Add a statement to the following program, on line 20, that will input a value from the file SCORE and assign that value to the variable X\$.

```
10 OPEN #5:"NAME=SCORE//3",DISPLAY,INPUT
30 S1$="SCORE = "
40 S2$=S1$&X$
50 PRINT #255:S2$
60 END
```

Answer: _____

Answers

Question 1

- a. Valid
- b. Invalid—too many characters
- c. Invalid—10 is not a simple filename
- d. Valid
- e. Invalid—name must start with a letter

Question 2

- c CLEAR DATA
- b SAVE
- a LIST
- d LOAD DATA

Question 3

30 PRINT #1:S\$

Question 4

20 LINPUT #5:X\$

Question 1

- a. 100%
- b. 100% for more than 100%
- c. 100% for a single firm
- d. 100%
- e. 100% for more than 100%

Question 2

- a. 100%
- b. 100%
- c. 100%
- d. 100%
- e. 100%

Question 3

- a. 100%

Question 4

- a. 100%

Chapter 2. INTERNAL files - sequential access

Introduction

In this chapter, you will learn how to process another type of data file. You will learn about internal files.

You will learn the statements that make internal files available to a program. You will also learn how to copy data into a file, and how to retrieve data from it.

You cannot load and list an internal file like you can a display file. Instead, you can only create and access the file while running a BASIC program.

Objectives

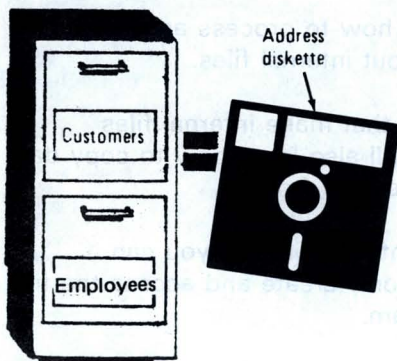
Upon completion of this chapter, you should be able to do the following:

- Open an internal file by using the OPEN statement.
- Copy data into a file by using the WRITE statement.
- Retrieve data from a file by using the READ statement.
- Specify the format of input or output data by using the FORM statement.

If you are familiar with these tasks, try the exercises at the end of this chapter. If not, read through the chapter before going on to the exercises.

INTERNAL files - sequential access

Creating a file



In this chapter, we're going to show you how to enter another data file of customer information (name and address).

However, in this chapter, we are going to use an INTERNAL file. You cannot enter an internal file at the keyboard with CLEAR DATA and SAVE.

Instead, you must write the data to a file, much like you printed data to a display file.



An internal file is divided into *records*. One record contains all of the information for each subject in the file.

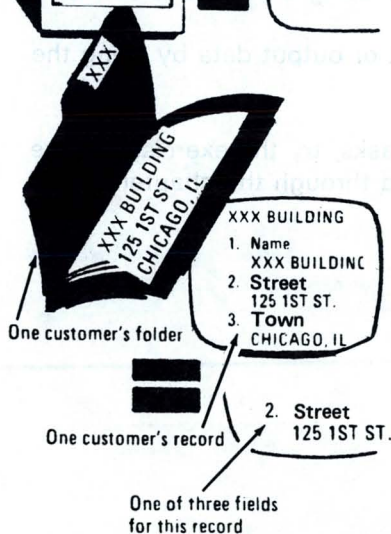
For example, in our file of customer information, one record contains one name and its corresponding address.

A diskette can be compared to a file cabinet. One file on a diskette is like one drawer of the file cabinet.

Within this file drawer are separate folders, each one containing information on a specific subject.

A *record* in a diskette file is like one of these folders in that everything on it is related and treated as one unit.

In the example to the left, both the folder and the record contain smaller pieces of information about the customer named XXX Building. Those pieces of information within that record are called *fields*.



Let's go over that again.

A diskette is like a file cabinet.

Each file on the diskette is like a file drawer in the cabinet.

Each record in the file is like a folder in the file drawer.

Each field in the record is like a piece of information in the folder.

There's one thing different about the records and fields on a diskette. That is, each record in a file must be the same length, but the fields in one record can be different lengths.

For example, look at this customer file.

Record 1	XXX BUILDING	125 1ST ST.	CHICAGO, IL
Record 2	XYZ PLUMBING	1830 5TH AVE.	CHICAGO, IL
	↑ NAME	↑ STREET	↑ TOWN

Both records are the same length. They are divided into fields of name (18 characters), street (19 characters), and town (11 characters).

INTERNAL files - sequential access

Creating a file (continued)

Now let's enter our customer file. Although we are only writing three items for each record, we will want to set a longer length than we actually need for the record. We are only writing the name and address now. But later, we may want to add more fields of information for each customer.

Let's assign a name field of 20 characters, a street field of 20 characters, and a town field of 20 characters. If we allow 67 characters for additional entries, each record will be 127 characters long ($20+20+20+67=127$).

All set? Then enter the following program:

```
CLEAR
10 OPTION BASE 1
20 DIM NAME$*20,STREET$*20,TOWN$*20
30 OPEN #1:"NAME=CUST//1,SIZE=0,RECL=127",INTERNAL,OUTPUT
40 PRINT "ENTER NAME"
50 LINPUT NAME$
60 IF NAME$="LAST" THEN GOTO 140
70 PRINT "ENTER STREET"
80 LINPUT STREET$
90 PRINT "ENTER TOWN"
100 LINPUT TOWN$
110 WRITE #1,USING 120:NAME$,STREET$,TOWN$
120 FORM POS 1,C 20,C 20,C 20
130 GOTO 40
140 END
```

Now run the program, and enter the responses shown. Your System/23 will know there are no more records to add to the file when you enter LAST for the name.

```
RUN
ENTER NAME

?XXX BUILDING
ENTER STREET
```

```
?125 1ST ST.
ENTER TOWN
```

```
?CHICAGO, IL
ENTER NAME
```

```
?XYZ PLUMBING
ENTER STREET
```

```
1830 5TH AVE.
ENTER TOWN
```

```
?CHICAGO, IL
ENTER NAME
```

```
?LAST
```

```
READY
```

```
INPUT
```

```
RUN
ENTER NAME
```

```
?XXX BUILDING
ENTER STREET
```

```
?125 1ST ST.
ENTER TOWN
```

```
?CHICAGO, IL
ENTER NAME
```

```
?XYZ PLUMBING
ENTER STREET
```

```
?1830 5TH AVE.
ENTER TOWN
```

```
?CHICAGO, IL
ENTER NAME
```

```
?LAST
```

After you enter LAST for the name, the program will end. (Remember that you must enter LAST, exactly as it is shown in line 60 of your program, that is, in all capital letters.) READY INPUT should now appear on the status line.

If your program didn't run, review the steps outlined on page 1-6 of this book. Then make any necessary corrections, and rerun the program.

INTERNAL files - sequential access

Creating a file (continued)

Let's look at the statements that open your file and write data to it.

First, let's look at the OPEN statement in line 30.

```
30 OPEN #1: "NAME=CUST//1, SIZE=0, RECL=127", INTERNAL, OUTPUT
```

As you learned in Chapter 1, OPEN makes a file available to a BASIC program.

The #1 is the *file reference* number. The number *must* be preceded by a pound sign (#).

If a data file is used by more than one program, its file reference can be different in each program. For example, CUST might be referred to by #1 in one program and #14 in another. But in any one program, its file reference must always be the same number.

```
30 OPEN #1: "NAME=CUST//1, SIZE=0, RECL=127", INTERNAL, OUTPUT
```

CUST//1 tells your System/23 which file to open. Because your diskette is on drive 1, you entered CUST//1.

If you are using drive 3, line 30 should look like this:

```
30 OPEN #1: "NAME=CUST//3, SIZE=0, RECL=127", INTERNAL, OUTPUT
```

```
30 OPEN #1: "NAME=CUST//1, SIZE=0, RECL=127", INTERNAL, OUTPUT
```

SIZE specifies the amount of space to be reserved on the diskette when a new file is created. It is required for new files, but it cannot be specified when opening a file that already exists. You must leave out SIZE in order to use a file a second time.

Remember from Chapter I that we will be using SIZE=0 in our examples. After your System/23 uses all the space it has reserved on a diskette for a file, it automatically reserves more space for that file.

```
30 OPEN #1:"NAME=CUST//1,SIZE=0,RECL=127",INTERNAL,OUTPUT
```

The RECL specifies the length of each record in the file. This is the total of the field lengths. In the file you created, it consists of:

Field	Number of characters
Name	20
Street	20
Town	20
Extra space	67
	127 total record length

The RECL *must* be specified for a new file and *cannot* be specified for an existing file.

```
30 OPEN#1:"NAME=CUST//1,SIZE=0,RECL=127",INTERNAL,OUTPUT
```

INTERNAL tells your System/23 that you are opening an *internal* data file. It can only be accessed through program control.

```
30 OPEN#1:"NAME=CUST//1,SIZE=0,RECL=127",INTERNAL,OUTPUT
```

OUTPUT specifies that you can only copy data into the file by using WRITE statements. This is like printing to a display file.

INTERNAL files - sequential access

Creating a file (continued)

Now let's look at the WRITE statement in line 110.

```
110 WRITE #1, USING 120:NAME$, STREET$, TOWN$
```

WRITE is a statement that adds a record to an internal file. Records are added to the end of a file, one item after another. Any fields in a record that do not receive a value are filled with blanks.

```
110 WRITE #1, USING 120:NAME$, STREET$, TOWN$
```

The #1 is the file reference number you assigned in line 30.

```
110 WRITE #1, USING 120:NAME$, STREET$, TOWN$
```

USING 120 tells your System/23 to write the data in the format specified in line 120. This is an optional entry. If you do not specify USING, the data in each record will be stored one item after the other, along with some control data that is supplied by your System/23. If you write data to a file without the USING, you can't read the data with a USING. We'll look at READ on the next few pages.

```
110 WRITE #1, USING 120:NAME$, STREET$, TOWN$
```

The values of NAME\$, STREET\$, and TOWN\$ are the values that are being written to the file. If you are writing more than one value to a file, you must separate the names with commas.

Before going on to the READ statement, save a copy of the program in the work area. Enter the following command:

```
SAVE WRITE.ADDRESS//1  
or  
SAVE WRITE.ADDRESS//3
```

Before we go on to the READ statement, let's run the WRITE.ADDRESS program again.

Let's add one more name and address to the file. Remember that we are writing records to a file with sequential access. Therefore, the records are added to the end of the file, one record after another.

First, list the program in the work area:

```
00010 OPTION BASE 1
00020 DIM NAME$*20,STREET$*20,TOWN$*20
00030 OPEN #1:"NAME=CUST//1,SIZE=0,RECL=127",INTERNAL,OUTPUT
00040 PRINT "ENTER NAME"
00050 INPUT NAME$
00060 IF NAME$="LAST" THEN GOTO 140
00070 PRINT "ENTER STREET"
00080 INPUT STREET$
00090 PRINT "ENTER TOWN"
00100 INPUT TOWN$
00110 WRITE #1,USING 120:NAME$,STREET$,TOWN$
00120 FORM POS 1,C 20,C 20,C 20
00130 GOTO 40
00140 END
```

LIST

Look at line 30. Notice that nothing is entered after the word OUTPUT. This tells your System/23 that we are using SEQUENTIAL access. We could include the word SEQUENTIAL, preceded by a comma, but it is not necessary, so we won't include it.

Before you can run this program again, you must delete SIZE=0 and RECL=127 from line 30. Enter:

```
30 OPEN #1:"NAME=CUST//1",INTERNAL,OUTPUT
```

List the new version of your program:

```
00010 OPTION BASE 1
00020 DIM NAME$*20,STREET$*20,TOWN$*20
00030 OPEN #1:"NAME=CUST//1",INTERNAL,OUTPUT
00040 PRINT "ENTER NAME"
00050 INPUT NAME$
00060 IF NAME$="LAST" THEN GOTO 140
00070 PRINT "ENTER STREET"
00080 INPUT STREET$
00090 PRINT "ENTER TOWN"
00100 INPUT TOWN$
00110 WRITE #1,USING 120:NAME$,STREET$,TOWN$
00120 FORM POS 1,C 20,C 20,C 20
00130 GOTO 40
00140 END
```

LIST

Now run your program again, and enter:

INTERNAL files - sequential access

Creating a file (continued)

```
RUN  
ENTER NAME
```

```
?ARTS AND CRAFTS  
ENTER STREET
```

```
?15 5TH AVE.  
ENTER TOWN
```

```
?NEW YORK, NY  
ENTER NAME
```

```
?LAST  
-
```

```
RUN  
ENTER NAME
```

```
?ARTS AND CRAFTS  
ENTER STREET
```

```
?15 5TH AVE.  
ENTER TOWN
```

```
?NEW YORK, NY  
ENTER NAME
```

```
?LAST
```

Remember that the word LAST tells your System/23 that you have added your last record.

Assigning values from a file

To retrieve data from the CUST file, you're going to change the program currently in the work area. List the program:

LIST

```
00010 OPTION BASE 1
00020 DIM NAME$*20,STREET$*20,TOWN$*20
00030 OPEN #1:"NAME=CUST//1",INTERNAL,OUTPUT
00040 PRINT "ENTER NAME"
00050 LINPUT NAME$
00060 IF NAME$="LAST" THEN GOTO 140
00070 PRINT "ENTER STREET"
00080 LINPUT STREET$
00090 PRINT "ENTER TOWN"
00100 LINPUT TOWN$
00110 WRITE #1,USING 120:NAME$,STREET$,TOWN$
00120 FORM POS 1,C 20,C 20,C 20
00130 GOTO 40
00140 END
```

Now make the following changes:

```
DEL 40,70
30 OPEN #1:"NAME=CUST",INTERNAL,INPUT
80 FOR I=1 TO 3
90 READ #1,USING 120:NAME$,STREET$,TOWN$
100 PRINT USING 120:NAME$,STREET$,TOWN$
110 NEXT I
DEL 130
```

List the new version of your program:

LIST

```
00010 OPTION BASE 1
00020 DIM NAME$*20,STREET$*20,TOWN$*20
00030 OPEN #1:"NAME=CUST",INTERNAL,INPUT
00080 FOR I=1 TO 3
00090 READ #1,USING 120:NAME$,STREET$,TOWN$
00100 PRINT USING 120:NAME$,STREET$,TOWN$
00110 NEXT I
00120 FORM POS 1,C 20,C 20,C 20
00140 END
```

INTERNAL files - sequential access

Assigning values from a file (continued)

Run the program:

RUN

```
RUN
XXX BUILDING      125 1ST ST.      CHICAGO, IL
XYZ PLUMBING      1630 5TH AVE.    CHICAGO, IL
ARTS AND CRAFTS   15 5TH AVE.      NEW YORK, NY
-
READY            INPUT
```

Let's look at the statements that opened your file and read data from it.

```
30 OPEN #1: "NAME=CUST", INTERNAL, INPUT
```

OPEN #1 has the same meaning as before. It opens a data file and assigns a file reference number of #1.

```
30 OPEN #1: "NAME=CUST", INTERNAL, INPUT
```

Just like in the OPEN statement in the WRITE.ADDRESS program, "NAME=CUST" indicates that the name of the file is CUST.

Notice that we did not include the RECL or SIZE in this statement, because CUST already exists. Also, we did not have to tell the System/23 which diskette drive the file is on. The System/23 searches the diskettes until it finds the CUST file.

```
30 OPEN #1: "NAME=CUST", INTERNAL, INPUT
```

This is an internal file. It is opened for input. That is, data can only be accessed by READ statements. READ is almost like INPUT, as we will show you.

Now let's look at the READ statement in line 90.

```
90 READ #1, USING 120:NAME$, STREET$, TOWN$
```

READ is a statement that assigns a value or values in a file to a specified variable or list of variables.

The #1 is the file reference number you assigned to CUST in line 30.

```
90 READ #1, USING 120:NAME$, STREET$, TOWN$
```

USING 120 tells your System/23 to read the data in the format specified in line 120. This is an optional entry. But, to read from a file without USING, the records must have been written to the file without USING.

```
90 READ #1, USING 120:NAME$, STREET$, TOWN$
```

NAME\$, STREET\$, and TOWN\$ are the variables in the program that are being assigned values from the file. The variable NAME\$ will receive the first value in the record. The variable STREET\$ will receive the second value in the record. The variable TOWN\$ will receive the third value in the record.

You are reading the records *sequentially*. That means that the records are read in the same order in which they were written. Therefore, the first time line 90 executes, the first record in the file is read. The second time line 90 executes, the second record is read, etc.

INTERNAL files - sequential access

Assigning values from a file (continued)

We knew that there were three records in the file, so we set up our FOR/NEXT loop to execute three times.

```
80 FOR I=1 TO 3
110 NEXT I
```

The variable names here must be character variable names, because you are reading character strings. If you were reading numeric values, you would enter numeric variable names.

The names of the variables that receive a value do not have to match the names of the values that were written to the file. For example, line 90 could look like:

```
90 READ #1,USING 120:N$,S$,T$
```

If you are reading more than one value from a file, you must separate the names with commas, as in NAME\$,STREET\$,TOWN\$ or N\$,S\$,T\$.

You can think of reading values from a file just as you read values with READ and DATA in Book II. A value in the file (or data list) is assigned to a variable in the program. The values in the file are like the values in DATA statements. But you can store *more* values in a file, and the values can be used by more than one program.

Before going on to the exercises, save a copy of the program in the work area. Enter the following:

```
SAVE READ.ADDRESS//1
or
SAVE READ.ADDRESS//3
```

Chapter summary

An internal file is created only through a program. It can be accessed sequentially. That is, records are written to the file and read from the file in order (record one, record two, record three, etc.).

To copy data into an internal file, you use the WRITE statement. The file must be opened for OUTPUT.

To retrieve data from an internal file, you use the READ statement. The file must be opened for INPUT.

Data is written to an internal file by records. One record pertains to one particular subject. Each record is divided into fields. A field is one piece of information about the subject.

Each record in an internal file must be the same length, but the fields in a record can be different lengths.

INTERNAL files - sequential access

Exercises

Question 1

Match the following statements with their corresponding functions.

- | | |
|---------|---|
| __WRITE | a. Makes an internal file available for input or output of data |
| __OPEN | b. Assigns values from a data file to variables in a program |
| __READ | c. Specifies format of input or output data records |
| __FORM | d. Adds data records to the end of an internal file |

Question 2

Match the parts of the following OPEN statement with their corresponding functions.

```
80 OPEN #1:"NAME=ACCOUNTS//1,SIZE=0,RECL=64",INTERNAL,OUTPUT
```

- | | |
|-----------|---|
| __RECL=64 | a. The file reference number used to identify the file throughout the program |
| __//1 | b. Identifies the diskette drive |
| __OUTPUT | c. Sets the record length for a new file |
| __#1 | d. Specifies that the file can only be accessed by WRITE statements |

Question 3

Assume you have run this program before you answer a, b, and c.

```
10 NAME$="JOHN DOE"  
20 OPEN #1:"NAME=XYZ.F1//1,SIZE=0,RECL=24",INTERNAL,OUTPUT  
30 WRITE #1,USING 40:NAME$  
40 FORM C 24  
50 END
```

a. How many records are there in the file XYZ.F1?

b. How many positions are there in each record?

c. What is in each position of record 1? (Use the dotted line to indicate positions.)

INTERNAL files - sequential access

Exercises (continued)

Question 4

What will appear on the screen if you run this program after running the program in Question 3?

```
10 OPEN #1:"NAME=XYZ.F1",INTERNAL,INPUT
20 READ #1,USING 40:A$
30 PRINT "THE NAME IS ";A$
40 FORM C 24
50 END
```

Answer: _____

Answers

Question 1

- d WRITE
- a OPEN
- b READ
- c FORM

Question 2

- c RECL=64
- b //1
- d OUTPUT
- a #1

Question 3

- a. 1
- b. 24
- c. JOHN DOE

Question 4

THE NAME IS JOHN DOE

Question 1

Answer 1
Option 1
Option 2
Option 3

Question 2

Answer 2
Option 1
Option 2
Option 3

Question 3

Answer 3
Option 1
Option 2
Option 3

Question 4

Answer 4
Option 1
Option 2
Option 3

Chapter 3. INTERNAL files - relative access

Introduction

In this chapter, you will learn how to access an individual record in a data file by entering its relative record number. This is known as *relative*, or direct, access.

When we use relative access with a file, we will refer to the file as a relative record I/O file. I/O stands for input and/or output.

Objectives

Upon completion of this chapter, you should be able to do the following:

- Open a relative record I/O file by using the OPEN statement.
- Copy data into a relative record I/O file by using the WRITE statement.
- Retrieve data from a relative record I/O file by using the READ statement.

If you are familiar with these tasks, try the exercises at the end of this chapter. If not, read through the chapter before going on to the exercises.

The OPEN statement

In Chapter 2, you learned how to open an internal file. You learned how to write data to a file and how to read data from a file.

The file you used in the last chapter was opened for sequential access. Records were written to the file one record after another. Any new records were always added to the end of the file.

When you read the data from your file, you started with the first record. You read the rest of the records in the same order that they were written to the file. (First record one, then record two, etc.)

In this chapter, you will learn how to access any record directly. To do so, you must open your file for relative access. Let's see how.

Do you remember the first program you wrote in the last chapter? It was WRITE.ADDRESS. This program opened your file and added records to it.

Since you have already entered most of what you'll need, we'll use this program again. First of all, you'll have to load your program back into the work area. Enter the following:

```
LOAD WRITE.ADDRESS
```


INTERNAL files - relative access

The WRITE statement

Do you remember what this program looks like? List it:

LIST

```
00010 OPTION BASE 1
00020 DIM NAME$*20,STREET$*20,TOWN$*20
00030 OPEN #1:"NAME=CUST//1,SIZE=0,RECL=127",INTERNAL,OUTPUT
00040 PRINT "ENTER NAME"
00050 LINPUT NAME$
00060 IF NAME$="LAST" THEN GOTO 140
00070 PRINT "ENTER STREET"
00080 LINPUT STREET$
00090 PRINT "ENTER TOWN"
00100 LINPUT TOWN$
00110 WRITE #1,USING 120:NAME$,STREET$,TOWN$
00120 FORM POS 1,C 20,C 20,C 20
00130 GOTO 40
00140 END
```

Now we're going to change the file CUST to a relative record I/O file. Enter the following:

```
30 OPEN #1:"NAME=CUST",INTERNAL,OUTPUT,RELATIVE
```

By entering RELATIVE in the OPEN statement, you have opened the file CUST for relative or direct access.

Look again at line 30. Do you remember why you didn't include the file size or record length in the file ID? It's because the file CUST already exists on your diskette.

Also, because the file CUST already exists, you didn't need the //1 (or//3).

INTERNAL files - relative access

The WRITE statement

When you write a record to a relative record I/O file, you must indicate the number of the record you are writing. So, you will have to make a few more changes to your program.

Change line 110 to the following statement:

```
110 WRITE #1,USING 120,REC=N:NAME$,STREET$,TOWN$
```

We are indicating the record to be written (REC=) with the variable N. We know that there are three records in the CUST file. So, the next record will be record number 4.

Let's put the statement N=4 on line 35. Enter:

```
35 N=4
```

Now, when line 110 is executed, N will equal 4. The next name and address will be written in the fourth record (REC=N and N=4).

Then, in order to have the next record after that be record number 5, enter the following statement:

```
125 N=N+1
```

List this new version of your program:

LIST

```
00010 OPTION BASE 1
00020 DIM NAME$*20,STREET$*20,TOWN$*20
00030 OPEN #1:"NAME=CUST",INTERNAL,OUTPUT,RELATIVE
00035 LET N=4
00040 PRINT "ENTER NAME"
00050 INPUT NAME$
00060 IF NAME$="LAST" THEN GOTO 140
00070 PRINT "ENTER STREET"
00080 INPUT STREET$
00090 PRINT "ENTER TOWN"
00100 INPUT TOWN$
00110 WRITE #1,USING 120,REC=N:NAME$,STREET$,TOWN$
00120 FORM POS 1,C 20,C 20,C 20
00125 LET N=N+1
00130 GOTO 40
00140 END
```

Now run your new program, and enter the responses shown. Remember that the DIM statement in line 20 limits the maximum number of characters in your entries to 20.

As in Chapter 2, your System/23 will know there are no more records to add to the file when you enter LAST for the name.

```
RUN
ENTER NAME

?SMITH INC.
ENTER STREET

?1000 1ST AVE.
ENTER TOWN

?NEW YORK, NY
ENTER NAME

?JONES LTD.
ENTER STREET

?3050 2ND AVE.
ENTER TOWN

?NEW YORK, NY
ENTER NAME

?LAST

-
READY          INPUT
```

```
RUN
ENTER NAME

?SMITH INC.
ENTER STREET

?1000 1ST AVE.
ENTER TOWN

?NEW YORK, NY
ENTER NAME

?JONES LTD.
ENTER STREET

?3050 2ND AVE.
ENTER TOWN

?NEW YORK, NY
ENTER NAME

?LAST
```

READY INPUT should now appear on your screen. Let's take a look at what happened when you ran this program.

INTERNAL files - relative access

The WRITE statement (continued)

First of all, you opened the file CUST, which already existed on the diskette in drive 1 (or 3). In this new program, however, you opened the file for relative access. A file that has been opened for sequential access in one program *can* be opened for relative access in another program.

Next, you wrote two records to the file. Both times you used the WRITE statement in line 110.

```
110 WRITE #1,USING 120,REC=N:NAME$,STREET$,TOWN$
```

The values that were written to the file were the values of NAME\$, STREET\$, and TOWN\$.

With relative access, you indicate the number (with REC=) of the record you are writing. Your file CUST should now contain the following records:

XXX BUILDING	125 1ST ST.	CHICAGO, IL
XYZ PLUMBING	1830 5TH AVE.	CHICAGO, IL
ARTS AND CRAFTS	15 5TH AVE.	NEW YORK, NY
SMITH INC.	1000 1ST AVE.	NEW YORK, NY
JONES LTD.	3050 2ND AVE.	NEW YORK, NY

Before we go on, let's save this new version of our program.

Do you remember how to save a program using its original filename? Enter the following:

REPLACE

REPLACE will always save your new program in the file with the name you entered when you said LOAD.

Assigning values from a file

Now we're going to look at the READ statement that is used with relative record I/O files. This time we're going to use the second program you wrote in the last chapter.

Do you remember the READ.ADDRESS program? It read the records in your CUST file and displayed the data on your screen.

To use READ.ADDRESS again, enter the following:

```
LOAD READ.ADDRESS
```

Now list the program:

```
LIST
```

```
00010 OPTION BASE 1
00020 DIM NAME$*20,STREET$*20,TOWN$*20
00030 OPEN #1:"NAME=CUST",INTERNAL,INPUT
00080 FOR I=1 TO 3
00090 READ #1,USING 120:NAME$,STREET$,TOWN$
00100 PRINT USING 120:NAME$,STREET$,TOWN$
00110 NEXT I
00120 FORM POS 1,C 20,C 20,C 20
00140 END
```

This time, we will have to make three changes to the program. First the file has to be opened again for relative access.

INTERNAL files - relative access

Assigning values from a file (continued)

Do you remember how to do this? Enter the following:

```
30 OPEN #1:"NAME=CUST",INTERNAL,INPUT,RELATIVE
```

Now, this time we are going to read just the second record. We have to take out the FOR/NEXT loop. Enter the following:

```
DEL 80  
DEL 110
```

And finally, we have to tell the System/23 to read only the second record in the CUST file. Enter the following. (Don't forget that you can scroll down to line 90 on your screen and insert ,REC=2.)

```
90 READ #1,USING 120,REC=2:NAME$,STREET$,TOWN$
```

Relative access is a good technique to use when you know the relative record number of the record you want to read.

List your program now:

```
LIST
```

```
00010 OPTION BASE 1  
00020 DIM NAME$*20,STREET$*20,TOWN$*20  
00030 OPEN #1:"NAME=CUST",INTERNAL,INPUT,RELATIVE  
00090 READ #1,USING 120,REC=2:NAME$,STREET$,TOWN$  
00100 PRINT USING 120:NAME$,STREET$,TOWN$  
00120 FORM POS 1,C 20,C 20,C 20  
00140 END
```

Now let's run the program and see what is displayed:

RUN

```

RUN
XYZ PLUMBING      1830 5TH AVE.      CHICAGO,IL
-

```

Only one name and address should appear on your screen.

This is the data that is stored in record number 2 of your CUST file.

A program using relative access reads whichever record is specified by REC= in the READ statement.

Your turn!

What would be displayed if line 90 looked like this?

```

90 READ #1,USING 120,REC=5:NAME$,STREET$,TOWN$

```

Answer: _____

This would be displayed on your screen:

```

JONES LTD.      3050 2ND AVE.      NEW YORK, NY

```

What happens if you open a file for relative access, and you don't specify the record to be read? It will read the next record.

Don't forget: To read a specific record from a file, you must include the word RELATIVE in your OPEN statement. You must also include REC= in your READ statements.

INTERNAL files - relative access

Chapter summary

Record I/O files can be accessed sequentially or directly. Relative, or direct, access allows you to read an individual record in a file by specifying its relative record number.

To allow relative access to a file, you must enter **RELATIVE** in the **OPEN** statement, as in:

```
OPEN #1: "NAME=A", INTERNAL, INPUT, RELATIVE
```

The **WRITE** statement adds records to a relative record I/O file by using a specific record number, as in:

```
WRITE #1, REC=3: A, B, C
```

To read a specific record from a relative record I/O file, you must enter the record number in the **READ** statement, as in:

```
READ #1, REC=3: A, B, C
```

A **READ** statement without **REC=** reads the record that follows the record previously read.

Exercises**Question 1**

Match the parts of the following OPEN statement with their corresponding functions:

80 OPEN #1:"NAME=ACCOUNTS//1",INTERNAL,INPUT,RELATIVE

- | | |
|--------------|--|
| ___ ACCOUNTS | a. The name of the file stored on diskette |
| ___ #1 | b. Specifies the file can be accessed by READ statements |
| ___ INPUT | c. Allows direct access to a specified record |
| ___ RELATIVE | d. The file reference number throughout the program |

Question 2

Assume that the internal file INVENT is stored on diskette drive 1. It contains the following records:

001	NUTS	100	.08
107	BOLTS	50	.15
115	SCREWS	50	.06

Write a statement that will open the file INVENT for relative access.

Answer: _____

INTERNAL files - relative access

Exercises (continued)

Question 3

Using the variables I\$, D\$, N, and C, and line numbers 10, 20, and 30, write a three-line program that will read record number 2 of the INVENT file.

Answer: _____

Question 4

Change the OPEN statement in the program in Question 3 to allow you to add another record to the INVENT file.

Answer: _____

Answers

Question 1

a ACCOUNTS

d #1

b INPUT

c RELATIVE

Question 2

```
10 OPEN #1:"NAME=INVENT//1",INTERNAL,INPUT,RELATIVE
```

You could have used any line number and any file reference number.

Question 3

```
10 OPEN #1:"NAME=INVENT//1",INTERNAL,INPUT,RELATIVE
20 READ #1,REC=2:I$,D$,N,C
30 END
```

Question 4

```
10 OPEN #1:"NAME=INVENT//1",INTERNAL,OUTPUT,RELATIVE
```

The //1 in questions 2, 3, and 4 is optional because the INVENT file already exists.

Question 1

a ACCOUNTS

b WIT

c INPUT

d RELATIVE

Question 2

10. The correct answer is (a) because the word "relative" is used in the same way as in the question.

You should also note that the word "relative" is used in the same way as in the question.

Question 3

11. The correct answer is (a) because the word "relative" is used in the same way as in the question.

The word "relative" is used in the same way as in the question.

The word "relative" is used in the same way as in the question.

Question 4

12. The correct answer is (a) because the word "relative" is used in the same way as in the question.

The word "relative" is used in the same way as in the question.

The word "relative" is used in the same way as in the question.

Chapter 4. Repositioning a file

Introduction

In this chapter, you will learn how to reposition a file. This means that you will be able to access the file again either at the beginning or at a specified record number.

We will discuss the two different access methods that you have used with internal files: sequential access and relative access.

You will learn how to reaccess a previously read data record and how to change a previously written data record.

You will also learn how to close an opened file and how to specify the action to be taken when an error occurs while accessing a file.

Objectives

Upon completion of this chapter, you should be able to do the following:

- Reposition a file by using the RESTORE statement.
- Assign a value from the record just previously read by using the REREAD statement.
- Update a record in a file by using the REWRITE statement.
- Remove a record from a file by using the DELETE statement.
- Close a file by using the CLOSE statement.
- Specify the action to be taken within a program when an error occurs while accessing a data file.

Repositioning a file

Introduction (continued)

If you are familiar with these tasks, try the exercises at the end of this chapter. If not, read through the chapter before going on to the exercises.

The RESTORE statement

Do you remember how you used the RESTORE statement in Book II? You entered RESTORE to go back to the beginning of your DATA values.

```
DATA 10,20,30
READ A,B,C
RESTORE
READ D,E,F
```

You can do the same thing with values stored in a file.

```
READ #1:A,B,C
RESTORE #1:
READ #1:D,E,F
```

In this case, you are going back to the beginning of a file. You are assigning the same values to D, E, and F that you assigned to A, B, and C.

The #1 in RESTORE #1 is the file reference number. If you do not include a file reference number, RESTORE refers to the values in your DATA statements.

You can also use the RESTORE statement in a program to rewrite data to a file after reading data from it. (REWRITE will be discussed in this chapter.)

```
READ #1:A,B,C
RESTORE #1:
READ #1:
REWRITE #1:A,B,C
```

Note: To read from and write to the same file in a program, the file must be opened with OUTIN in the OPEN statement. Or, you must close the file after reading from it and open it again when you want to write to it.

Repositioning a file

The RESTORE statement (continued)

Do you remember the two programs that you saved in Chapter 2? Both programs used sequential access. One program added data to a file. The other program read data from a file. Next we'll show you how you can do both in the same program.

Keep in mind, as you look at and run the next program, that we are using *sequential* access in this file. Remember: That means we are reading the records in the same order in which they are written.

Let's write a program to store employee data. Each record will pertain to one employee. The fields in each record will be name, sex, date of birth, and date of hire.

We will close the file and reopen it to read data from the file we just created and print that data in a report. Once we have printed the report, we will use the RESTORE statement to restore the file back to the beginning. This is done to read the name and print a second report, using only the name of the employees.

The rest of the statements in this program should be familiar to you. If you are confused by a particular section, you may want to go back and review an earlier part of this course. For example, the INPUT statement and FOR/NEXT loop are covered in Book II. The PRINT and PRINT #255 statements are covered in Book III.

All set? OK, let's enter the program from the next page.


```
CLEAR
00010 OPTION BASE 1
00020 DIM NAME$*20,SEX$*1
00030 OPEN #1:"NAME=EMP//1,SIZE=0,RECL=40",INTERNAL,OUTPUT
00040 PRINT "ENTER NAME"
00050 INPUT NAME$
00060 IF NAME$="LAST" THEN 170
00070 PRINT "ENTER SEX (M OR F)"
00080 INPUT SEX$
00090 PRINT "ENTER BIRTH DATE (MO,DAY,YR)"
00100 INPUT M1,D1,Y1
00110 PRINT "ENTER HIRE DATE (MO,DAY,YR)"
00120 INPUT M2,D2,Y2
00130 LET COUNT=COUNT+1 ! NUMBER OF EMPLOYEES
00140 WRITE #1,USING 150:NAME$,SEX$,M1,D1,Y1,M2,D2,Y2
00150 FORM POS 1,C 20,C 1,6*N 2
00160 GOTO 40
00170 CLOSE #1:
00180 OPEN #1:"NAME=EMP",INTERNAL,INPUT
00190 FOR I=1 TO COUNT
00200 READ #1,USING 150:NAME$,SEX$,M1,D1,Y1,M2,D2,Y2
00210 PRINT #255,USING 250:NAME$,SEX$
00220 PRINT #255,USING 260:"BIRTH:",M1,D1,Y1
00230 PRINT #255,USING 260:"HIRE:",M2,D2,Y2
00240 NEXT I
00250 FORM SKIP 2,C 20,X 3,C 1
00260 FORM C 8,N 2,X 1,N 2,X 1,N 2
00270 RESTORE #1:
00280 FOR I=1 TO COUNT
00290 READ #1,USING 150:NAME$
00300 PRINT #255,USING 320:NAME$
00310 NEXT I
00320 FORM SKIP ,C 20
00330 END
```

Repositioning a file

The RESTORE statement (continued)

Before you run this program, save a copy of it.
Enter:

```
SAVE EMP.INFO//1  
or  
SAVE EMP.INFO//3
```

If your program doesn't run, check first to make sure that you didn't make any typing errors. Then look up the action code and error code in your *Messages* manual to find out the exact cause of your problem.

Remember to expect the line of asterisks on your screen after you press Error Reset.

List the number of the line on which the program stopped. That number will be to the right of the error code on the status line.

If your error was in that line, fix it, then type GO.

If the line you fixed came before the line on which the program stopped, you will have to type GO END.

Fix the appropriate line and run the program again.

If the action code indicated a filename problem, enter FREE EMP to release the file the program created when it tried to run before. Then run the program again.

Use the following input data:

```
RUN
ENTER NAME
?F.L. BROWN
ENTER SEX (M or F)
?M
ENTER BIRTH DATE (MO, DAY, YR)
?6, 6, 38
ENTER HIRE DATE (MO, DAY, YR)
?3, 12, 69
ENTER NAME
?S.S. BLACK
ENTER SEX (M or F)
?F
ENTER BIRTH DATE (MO, DAY, YR)
?1, 4, 52
ENTER HIRE DATE (MO, DAY, YR)
?7, 16, 79
ENTER NAME
?F.R. GREEN
ENTER SEX (M or F)
?M
ENTER BIRTH DATE (MO, DAY, YR)
?10, 13, 48
ENTER HIRE DATE (MO, DAY, YR)
?1, 15, 73
ENTER NAME
?LAST
```

```
F.L. BROWN      M
BIRTH:  6  6 38
HIRE:   3 12 69

S.S. BLACK     F
BIRTH:  1  4 52
HIRE:   7 16 79

F.R. GREEN     M
BIRTH: 10 13 48
HIRE:   1 15 73

F.L. BROWN
S.S. BLACK
F.R. GREEN
```

The report shown at the left should be printed or displayed, depending on how you entered lines 210-300.

Repositioning a file

The RESTORE statement (continued)

The RESTORE statement in line 270 repositions the EMP file back to the beginning. Then you read the records in the same order in which they were written, beginning with record 1.

Now let's look at a program that uses relative access with RESTORE.

You're going to enter an inventory program.

This program is a short version of the complete inventory program found in Book VII. It may look like a lot of lines to enter, but you should be getting used to longer programs by now. All set? OK. Take your time and enter the following:

```
CLEAR
10 REM THIS PROGRAM READS AND WRITES RECORDS
20 DIM D$*20
30 OPEN #1:"NAME=ITEMS//1,SIZE=0,RECL=64",INTERNAL,OUTIN,RELATIVE
40 N1=0
50 PRINT "ENTER ITEM NUMBER"
60 INPUT I$
70 IF I$="LAST" THEN GOTO 180
80 N1=N1+1
90 PRINT "ENTER DESCRIPTION"
100 INPUT D$
110 PRINT "ENTER QUANTITY ON HAND"
120 INPUT Q
130 PRINT "ENTER UNIT COST"
140 INPUT P
150 WRITE #1,USING 160,REC=N1:I$,D$,Q,P,Q*P
160 FORM C 5,C 20,N 6,N 9.2,N 17.2
170 GOTO 50
180 RESTORE #1,REC=2: ! RESTORE FILE TO SECOND RECORD
190 READ #1,USING 160:I$,D$,Q,P,P1
200 REM READ RECORD IN THE FILE
210 PRINT USING 220:I$,D$,Q,P,P1
220 FORM C 6,C 20,PIC(###),X 4,PIC(####.##),X 3,PIC(####.##)
230 END
```

Repositioning a file

The RESTORE statement (continued)

Now run the program. Use the following input data:

```
RUN
ENTER ITEM NUMBER
?00001
ENTER DESCRIPTION
?NUTS
ENTER QUANTITY ON HAND
?2000
ENTER UNIT COST
?.29
ENTER ITEM NUMBER
?00002
ENTER DESCRIPTION
?BOLTS
ENTER QUANTITY ON HAND
?1500
ENTER UNIT COST
?.39
ENTER ITEM NUMBER
?LAST
```

```
RUN
ENTER ITEM NUMBER
?00001
ENTER DESCRIPTION
?NUTS
ENTER QUANTITY ON HAND
?2000
ENTER UNIT COST
?.29
ENTER ITEM NUMBER
?00002
ENTER DESCRIPTION
?BOLTS
ENTER QUANTITY ON HAND
?1500
ENTER UNIT COST
?.39
ENTER ITEM NUMBER
?LAST
00002 BOLTS          1,500      $0.39      $585.00
-
```

The data for the second record should have been displayed. If your program didn't run, go back and follow the steps on page 1-6 of this book.

Let's look again at the program that produced this report.

```
10 REM THIS PROGRAM READS AND WRITES RECORDS
20 DIM D$*20
30 OPEN #1:"NAME=ITEMS//1,SIZE=0,RECL=64",INTERNAL,OUTIN,RELATIVE
40 N1=0
50 PRINT "ENTER ITEM NUMBER"
60 INPUT I$
70 IF I$="LAST" THEN GOTO 180
80 N1=N1+1
90 PRINT "ENTER DESCRIPTION"
100 INPUT D$
110 PRINT "ENTER QUANTITY ON HAND"
120 INPUT Q
130 PRINT "ENTER UNIT COST"
140 INPUT P
150 WRITE #1,USING 160,REC=N1:I$,D$,Q,P,Q*P
160 FORM C 5,C 20,N 6,N 9.2,N 17.2
170 GOTO 50
180 RESTORE #1,REC=2: ! RESTORE FILE TO SECOND RECORD
190 READ #1,USING 160:I$,D$,Q,P,P1
200 REM READ RECORD IN THE FILE
210 PRINT USING 220:I$,D$,Q,P,P1
220 FORM C 6,C 20,PIC(##,###),X 4,PIC(####.##),X 3,PIC(#####.##)
230 END
```

Do you know what each statement does?

Repositioning a file

The RESTORE statement (continued)

Let's look at the statements that pertain to your data file.

Line 30 creates a new file on diskette drive 1. It is called ITEMS. If you are using drive 3, it would read ITEMS//3.

This is an internal file. It will be accessed by relative record number, with a record length of 64. It can be used for both output and input.

Line 150 writes five fields of data in each record of the file. When N1 equals 1, the data goes in the first record. When N1 equals 2, the data goes in the second record.

Line 180 repositions the file to record number 2. Record number 2 will be the next record to be accessed by READ.

If you don't tell your System/23 which record to restore (as in RESTORE #1:), the file will be repositioned at the beginning. Remember that if you are using sequential access, the RESTORE statement also repositions the file at the beginning.

Line 190 reads five items from the second record in the file.

We have shown you how to reposition a file so that you can access any record. Next we'll show you how to reaccess only the record that was just read.

The REREAD statement

Suppose you had run the inventory program and entered 100 items. Now you need to know what you entered for BOLTS. How can you find that information if you don't know its record number?

We'll show you how to read only the second field from each record. After you locate BOLTS in the file, you will read that entire record. Enter the following without using the AUTO command:

```
CLEAR
10 REM SEARCH FOR BOLTS
20 DIM D$*20
30 OPEN #1:"NAME=ITEMS",INTERNAL,INPUT,RELATIVE
40 FOR I=1 TO 100
50 READ #1,USING 55,REC=I:D$
55 FORM X 5,C 5
60 FORM C 5,C 20,N 6,N 9.2,N 17.2
70 IF D$ <> "BOLTS" THEN GOTO 110
80 REREAD #1,USING 60:I$,D$,Q,P,P1
90 PRINT USING 60:I$,D$,Q,P,P1
100 GOTO 120
110 NEXT I
120 END
```

Look at lines 50 and 55.

```
50 READ #1,USING 55,REC=I:D$
55 FORM X 5,C 5
```

These statements tell your System/23 to read only one field from the record. The field to be read is five characters long, and it begins in the sixth position in the record. Run the program:

RUN

```
RUN
00002BOLTS          1500      .39      585.00
```

Repositioning a file

The REREAD statement (continued)

Let's look at what else this program is doing.

Line 40 sets up a FOR-NEXT loop with values from 1 to 100. As we run the program, we will break out of the loop after finding BOLTS. This may happen before I=100.

That's all right. It will not cause an error. You can break out of any loop with a GOTO statement, even if you haven't completed the loop the number of times specified.

Line 50 reads characters in positions 6-10 from each record. It reads them using the FORM statement in line 55. By reading only one data item from each record, we save the time it would take to also read the unnecessary values.

We could have read the entire 20-character field for D\$. If we had, line 70 would need to check D\$(1:5). Remember from Book V that D\$(1:5) means the first five characters of D\$. If we look at all of D\$, it will look like this:
"BOLTS

Line 80 *rereads* the same record that was just read. But this time, the program is reading five fields, not just one.

REREAD acts like READ. It assigns values from a file to variables in a program. However, the record being reread is the same record that was just read.

Notice in lines 60 and 70 that you can execute other statements between the READ and REREAD.

The REWRITE statement

How do you update or change one of the records in a file? You use the REWRITE statement.

Let's look at an example. In the first record of ITEMS, you entered a quantity on hand of 2000. Suppose you sold 200. Your new quantity is 1800.

Let's write a program to update the record. Enter the following without using the AUTO command:

```
CLEAR
10 REM PROGRAM TO UPDATE 1ST RECORD
20 DIM D$*20
30 OPEN #1:"NAME=ITEMS",INTERNAL,OUTIN,RELATIVE
40 READ #1,USING 50,REC=1:Q,P
50 FORM X 25,N 6,N 9.2
60 Q=Q-200
70 REWRITE #1,USING 75:Q,Q*P
75 FORM X 25,N 6,X 9,N 17.2
80 PRINT "RECORD REWRITTEN"
90 END
```

Now run the program:

```
RUN
RECORD REWRITTEN
-
```

RUN

The REWRITE statement in line 70 tells your System/23 to copy data into the same record that was just read.

The fields that are skipped when the record is read and rewritten (by using X in the FORM statements) are left undisturbed.

As with REREAD, you can execute other statements between READ and REWRITE.

Repositioning a file

The DELETE statement

You already know how to remove a line from a program. DEL 70 deletes line 70. You can also delete records from a data file. In order to delete records from a data file, the file must be opened with OUTIN in the OPEN statement.

Suppose you sold the last bolt in your inventory. You are no longer going to stock that item, so you want to remove it from your file. Enter the following:

```
40 READ #1,USING 50,REC=2:
60 DELETE #1:
80 PRINT "RECORD DELETED"
DEL 70,75
```

The DELETE statement in line 60 removes a record from the file. It deletes the record that was just read.

Can you guess what this statement will do?

```
60 DELETE #1,REC=2:
```

It will delete record number 2 from file #1. If you do not tell your System/23 which record to delete, the last record read is deleted.

List the new version of your program:

```
LIST
```

Remember: When writing, reading, and deleting records, you can specify REC= *only* if the file has been opened for RELATIVE access.

```
00010 REM PROGRAM TO UPDATE 1ST RECORD
00020 DIM D$*20
00030 OPEN #1:"NAME=ITEMS",INTERNAL,OUTIN,RELATIVE
00040 READ #1,USING 50,REC=2:
00050 FORM X 25,N 6,N 9.2
00060 DELETE #1:
00080 PRINT "RECORD DELETED"
00090 END
```

The CLOSE statement

The CLOSE statement does the opposite of the OPEN statement. You use the CLOSE statement to deactivate a file while running a program.

To close the file ITEMS in the program in the work area, enter the following:

```
65 CLOSE #1:
```

Now run the program:

```
RUN  
RECORD DELETED  
-
```

```
RUN
```

When READY appears, you know that the second record has been deleted (line 60), and the file has been closed (line 65).

CLOSE is especially helpful when you are running a long program, or when you want to remove a diskette. A file will automatically be closed when your program reaches an END or STOP statement. The commands LOAD and CLEAR also close a file.

You can also use the CLOSE statement to free a file while you are running a program. For example, if you had wanted to free the ITEMS file after you finished using it, line 65 would have looked like this:

```
65 CLOSE #1,FREE:
```

Repositioning a file

Controlling file access errors

So far, you have learned to process files with these statements:

OPEN
READ/REREAD
WRITE/REWRITE
RESTORE
DELETE
CLOSE

An error could occur while performing any of these. You can include statements to control your programs when an error occurs while accessing a file. By doing so, you may save yourself the time and trouble of reentering or rerunning a program.

Some common error conditions and actions follow. They can be used with any of the file processing statements listed above.

One or more error conditions can be entered in a statement that accesses a file. You can use error conditions with READ, WRITE, OPEN, DELETE, etc.

When you use an error condition in a statement, you include the condition followed by a line number or label. If the condition occurs, control will branch to that line reference. If the condition does not occur, the line reference will be ignored. For a complete list of error conditions, see "EXIT" in the *BASIC Language Reference*.

Here are some of the error conditions you might receive.

CONV—A conversion error. The variable name doesn't match the type of data (numeric versus character), or the record is not long enough to hold all the items being written.

EOF—End of file. There are no more records in the file.

IOERR—Input/output error. An error has prevented completion of the statement which is not one of the other error conditions.

NOREC—No such record. The record number specified does not exist; it is less than or equal to 0; it has been deleted, or the number is greater than the number of records in the file.

SOFLOW—String overflow. The output field is not long enough for the character string being written, or the variable is not dimensioned large enough to hold the data being read.

Here are some example program statements using error conditions:

Repositioning a file

Controlling file access errors (continued)

```
10 OPEN #1:"NAME=ACC.RCVBL",INTERNAL,OUTIN,RELATIVE IOERR 40
20 GOTO 50
.
.
.
40 PRINT "FILE DOES NOT EXIST"
45 STOP
50 READ #2,REC=45:A,B,C CONV 120,NOREC 180,EOF 9999
.
.
.
180 PRINT "THERE IS NO RECORD 45"
```

You can also direct program control with one EXIT statement:

```
30 WRITE #5,REC=50:A$,B,C EXIT 70
70 EXIT CONV 120,NOREC 80,SOFLOW 300
```

If an error occurs in line 30, the action is determined by line 70.

Let's look at an example in a program. First of all, load the EMP.INFO program that you saved earlier in this chapter. Do you remember how? Enter:

```
LOAD EMP.INFO
```

List your program:

```
LIST
```

Repositioning a file

Controlling file access errors (continued)

```
00010 OPTION BASE 1
00020 DIM NAME$*20,SEX$*1
00030 OPEN #1:"NAME=EMP//1,SIZE=0,RECL=40",INTERNAL,OUTPUT
00040 PRINT "ENTER NAME"
00050 INPUT NAME$
00060 IF NAME$="LAST" THEN 170
00070 PRINT "ENTER SEX (M OR F)"
00080 INPUT SEX$
00090 PRINT "ENTER BIRTH DATE (MO,DAY,YR)"
00100 INPUT M1,D1,Y1
00110 PRINT "ENTER HIRE DATE (MO,DAY,YR)"
00120 INPUT M2,D2,Y2
00130 LET COUNT=COUNT+1 ! NUMBER OF EMPLOYEES
00140 WRITE #1,USING 150:NAME$,SEX$,M1,D1,Y1,M2,D2,Y2
00150 FORM POS 1,C 20,C 1,6*N 2
00160 GOTO 40
00170 CLOSE #1:
00180 OPEN #1:"NAME=EMP",INTERNAL,INPUT
00190 FOR I=1 TO COUNT
00200 READ #1,USING 150:NAME$,SEX$,M1,D1,Y1,M2,D2,Y2
00210 PRINT #255,USING 250:NAME$,SEX$
00220 PRINT #255,USING 260:"BIRTH:",M1,D1,Y1
00230 PRINT #255,USING 260:"HIRE:",M2,D2,Y2
00240 NEXT I
00250 FORM SKIP 2,C 20,X 3,C 1
00260 FORM C 8,N 2,X 1,N 2,X 1,N 2
00270 RESTORE #1:
00280 FOR I=1 TO COUNT
00290 READ #1,USING 150:NAME$
00300 PRINT #255,USING 320:NAME$
00310 NEXT I
00320 FORM SKIP ,C 20
00330 END
```

Now make the following changes:

```
30 OPEN #1:"NAME=EMP//1,SIZE=0,RECL=40",INTERNAL,OUTPUT IOERR 340
DEL 190
200 READ #1,USING 150:NAME$,SEX$,M1,D1,Y1,M2,D2,Y2 EOF 270
240 GOTO 200
DEL 280
290 READ #1,USING 150:NAME$ EOF 330
310 GOTO 290
330 STOP
340 PRINT "FILE ALREADY EXISTS"
350 PRINT "USE DIFFERENT OPEN STATEMENT"
360 OPEN #1:"NAME=EMP",INTERNAL,OUTPUT
370 GOTO 40
380 END
```

The IOERR in line 30 sends control to line 340 if the EMP file already exists. Instead of stopping your program with an error, we open the file correctly in line 360, and then control branches back to line 40.

The EOF in line 200 sends control to line 270. The EOF condition occurs when the READ is issued and all the records in the file have been read.

Go ahead and run the program with this input:

Repositioning a file

Controlling file access errors (continued)

```
RUN
FILE ALREADY EXISTS
USE DIFFERENT OPEN STATEMENT
ENTER NAME
```

```
?J.T. WHITE
ENTER SEX (M OR F)
```

```
?M
ENTER BIRTH DATE (MO,DAY,YR)
```

```
?3,16,47
ENTER HIRE DATE (MO,DAY,YR)
```

```
?3,16,70
ENTER NAME
```

```
?LAST
-
```

```
RUN
FILE ALREADY EXISTS
USE DIFFERENT OPEN STATEMENTS
ENTER NAME
?J.T. WHITE
ENTER SEX (M or F)
?M
ENTER BIRTH DATE (MO,DAY,YR)
?3,16,47
ENTER HIRE DATE (MO,DAY,YR)
?3,16,70
ENTER NAME
?LAST
```

```
F.L. BROWN      M
BIRTH:   6  6 38
HIRE:    3 12 69
```

```
S.S. BLACK      F
BIRTH:    1  4 52
HIRE:    7 16 79
```

```
F.R. GREEN      M
BIRTH:   10 13 48
HIRE:    1 15 73
```

```
J.T. WHITE      M
BIRTH:    3 16 47
HIRE:    3 16 70
```

```
F.L. BROWN
```

```
S.S. BLACK
```

```
F.R. GREEN
```

```
J.T. WHITE
```

The report shown at the left should be printed or displayed.

Notice that all of the records in this file were listed because the file was read from the beginning to the end.

Before we go on, let's save this new version of our program. Enter:

```
REPLACE
```

Chapter summary

The RESTORE statement repositions a file to the beginning or to a specified record.

The REREAD statement assigns values from the record just previously read. The REWRITE statement copies data into a previously written record.

The DELETE statement removes a record from a file. The CLOSE statement closes a file during program execution.

The EXIT statement directs program control when a file access error occurs. Error conditions include:

CONV—conversion

EOF—end of file

IOERR—input/output error

NOREC—no record found

SOFLOW—string overflow

Repositioning a file

Exercises

Assume that the file NAME.ADDR is stored on diskette drive 1. It contains the following records:

00001	ABC BUILDING	125 1ST ST	12
00103	XYZ PLUMBING	1830 5TH AVE	50
12345	AIRCO SALES	845 MAIN ST	50

Question 1

What will be displayed if you run the following programs?

a. 10 OPEN #1:"NAME=NAME.ADDR//1,SIZE=0",INTERNAL,INPUT
20 READ #1,USING 30:A\$
30 FORM C 5,X,C 28,N 2
40 PRINT A\$
50 END

Answer: _____

b. 10 OPEN #1:"NAME=NAME.ADDR",INTERNAL,INPUT
20 READ #1,USING 30:A\$
30 FORM C 5,X,C 28,N 2
40 REREAD #1,USING 30:B\$
50 PRINT A\$;B\$
60 END

Answer: _____

Question 2

What will be displayed if you run the following programs?

a. 10 OPEN #1:"NAME=NAME.ADDR", INTERNAL, INPUT, RELATIVE
20 READ #1, USING 50, REC=2:A\$
30 RESTORE #1:
40 READ #1, USING 50:B\$
50 FORM C 6
60 PRINT A\$;B\$
70 END

Answer: _____

b. 10 OPEN #2:"NAME=NAME.ADDR", INTERNAL, OUTIN, RELATIVE
15 DIM B\$*28
20 READ #2, USING 30, REC=3:A\$, B\$, CO
30 FORM C 5, X, C 28, N 2
40 CO=10
50 REWRITE #2, USING 30:A\$, B\$, CO
60 RESTORE #2:
70 READ #2, USING 30:A\$, B\$, CO EOF 100
80 PRINT USING 30:A\$, B\$, CO
90 GOTO 70
100 END

Answer: _____

Repositioning a file

Exercises (continued)

Question 3

What will be displayed if you run the following programs?

a. 10 OPEN #2:"NAME=NAME.ADDR",INTERNAL,INPUT
15 DIM B\$*28,E\$*28
20 DATA 00008,NEW PRODUCTS 100 1ST ST, 8
30 READ A\$,B\$,CO
40 READ #2,USING 50:D\$,E\$,CO1
50 FORM C 5,X,C 28,N 2
60 PRINT USING 50:A\$,B\$,CO
70 PRINT USING 50:D\$,E\$,CO1
80 END

Answer: _____

b. 10 OPEN #2:"NAME=NAME.ADDR",INTERNAL,OUTIN,RELATIVE
15 DIM B\$*28
20 DELETE #2,REC=2:
30 RESTORE #2:
40 READ #2,USING 50:A\$,B\$,CO EOF 70
50 FORM C 5,X,C 28,N 2
60 GO TO 40
70 PRINT A\$
80 END

Answer: _____

Answers

Question 1

- a. An error message—you cannot state the SIZE when opening an existing file.
- b. 0000100001

Question 2

- a. 00103 00001
- b. 00001 ABC BUILDING 125 1ST ST 12
00103 XYZ PLUMBING 1830 5TH AVE 50
12345 AIRCO SALES 845 MAIN ST 10

Question 3

- a. 00008 NEW PRODUCTS 100 1ST ST 8
00001 ABC BUILDING 125 1ST ST 12
- b. 12345

Question 1

1. All of the above—you can't tell from the data when
- opening an existing file.

Question 2

2. The correct answer is C. The correct answer is C.
The correct answer is C. The correct answer is C.
The correct answer is C. The correct answer is C.

Question 3

3. The correct answer is B. The correct answer is B.
The correct answer is B. The correct answer is B.

Chapter 5. INTERNAL files - key-indexed access

Introduction

In the first four chapters of this book, you learned about files. You learned how to access an individual record by entering its record number.

In this chapter, you will learn how to use a key-indexed file. You will learn how to access an individual record, even when you don't know its record number.

Objectives

Upon completion of this chapter, you should be able to do the following:

- Create an index file by using the LINK command and the INDEX Customer Support Function.
- Activate a key-indexed file by using the OPEN statement.
- Retrieve data from a key-indexed file by using the READ statement.
- Copy data into a key-indexed file by using the WRITE and REWRITE statements.
- Remove a record from a key-indexed file by using the DELETE statement.

If you are familiar with these tasks, try the exercises at the end of this chapter. If not, read through the chapter before going on to the exercises.

INTERNAL files - key-indexed access

What is a key-indexed file?

A *key-indexed file* is a record I/O file with an associated *index file*. The index file contains a *key* and relative record number for each record in the data file.

A key is a continuous set of characters used to identify each record. A key can be a customer number, an item number, or any other field in a record. It can be from 1 to 28 characters long, and it is read as a character string.

You must use the same location of characters in each record as the key. For example, the key could be the first five characters of each record. Or, it could be the characters in positions 10-30 of each record. You decide which field to use in each file.

Let's look at an example. In the following file of customer data, positions 1-3 in each record will be the key.

001	ABC BUILDING	125 1ST ST.
108	XYZ PLUMBING	1830 5TH AVE.
057	WORLDWIDE MFG	1000 8TH ST.

Data file

001	ABC BUILDING	125 1ST ST.
108	XYZ PLUMBING	1830 5TH AVE.
057	WORLDWIDE MFG	1000 8TH ST.

Index file

001	1
057	3
108	2

The index for this file will contain the following:

001	1
057	3
108	2

↑ ↑
Key Record number

Note: When you create an index for a file, the keys will be sorted in ascending order (001, 057, 108). The data file itself is unchanged.

When you want to access the data record for customer 108, you specify KEY="108". Your System/23 will search through the index until it finds key 108. It will see that key 108 is in the second record of the data file. Using record number 2, it directly accesses the correct record in the data file.

Key-indexed files are used when you want to access individual records directly. Instead of searching an entire data file, your System/23 searches the shorter index file.

If our file of customer data had 1000 records, it would be much faster to search an index file for a particular customer number, rather than to search the entire data file.

As you can see in our example, the keys do not have to be the same numbers as the relative record numbers. In fact, the key can be, and often is, alphabetic.

For example, in the following file, the key could be the town, the zip code, or the first three letters of the name.

GENERAL SYSTEMS	4111	NORTHSIDE	ATLANTA, GA	30327
OFFICE PRODUCTS	400	PARSONS POND	FRANKLIN LAKES, NJ	07417
GENERAL BUSINESS	1133	WESTCHESTER	WHITE PLAINS, NY	10604

The records in the data file can be added in any order, regardless of the keys. When you create an index, the keys are automatically sorted in ascending order.

INTERNAL files - key-indexed access

Setting up a key-indexed file

To set up an index, first you need a data file. Let's enter a new ITEM file and then create an index to go with it.

Enter the following:

```
CLEAR
10 REM PROGRAM TO CREATE MASTER DATA FILE
20 DIM D$*20
30 OPEN #1:"NAME=ITEM//1,SIZE=0,RECL=60",INTERNAL,OUTPUT
40 PRINT "ENTER ITEM NUMBER"
50 INPUT I$
60 IF I$="LAST" THEN GOTO 130
70 PRINT "ENTER DESCRIPTION,QUANTITY,COST"
80 INPUT D$,Q,P
90 REM WRITE ITEM TO FILE
100 WRITE #1,USING 110:I$,D$,Q,P,Q*P
110 FORM C 5,C 20,N 6,N 9.2,N 17.2
120 GOTO 40
130 END
```

Now run the program and enter the responses from the following page:

RUN
ENTER ITEM NUMBER
? 00001
ENTER DESCRIPTION, QUANTITY, COST
? NUTS, 1800, .29
ENTER ITEM NUMBER
? 00002
ENTER DESCRIPTION, QUANTITY, COST
? BOLTS, 1500, .39
ENTER ITEM NUMBER
? 12345
ENTER DESCRIPTION, QUANTITY, COST
? SCREWS, 850, .11
ENTER ITEM NUMBER
? 10008
ENTER DESCRIPTION, QUANTITY, COST
? NAILS, 5000, .02
ENTER ITEM NUMBER
? LAST

The file ITEM now has four records.

00001	NUTS	1800	.29	522.00
00002	BOLTS	1500	.39	585.00
12345	SCREWS	850	.11	93.50
10008	NAILS	5000	.02	100.00

INTERNAL files - key-indexed access

Creating an index file

Now we're going to create an index for the data file ITEM. First of all, you will need the Customer Support Functions diskette that contains INDEX. This diskette was supplied by IBM when you received your System/23.

Insert your Customer Support Functions diskette containing INDEX in a diskette drive.

You will have to replace the diskette you're using for this course with the Customer Support Functions diskette if you only have one diskette drive.

Once your diskette is in place, enter the following:

LINK INDEX

INDEX is a program on your Customer Support Functions diskette. It creates an index file for any master data file. In our example, the master data file is ITEM.

LINK is a command that loads and runs a Customer Support Function.

When INDEX creates an index file, several screens will be displayed. Enter your responses exactly as shown on the screens following on the pages.

The following OPTION MENU should appear on your screen. You may now remove the Customer Support Functions diskette and reinsert yours if necessary.

01. CSE	02. CSE	03. CSE	04. CSE	05. CSE	06. CSE
07. CSE	08. CSE	09. CSE	10. CSE	11. CSE	12. CSE
13. CSE	14. CSE	15. CSE	16. CSE	17. CSE	18. CSE
19. CSE	20. CSE	21. CSE	22. CSE	23. CSE	24. CSE

Create Index File Menu 04-010

Choose one of the following:

- 1. Description of Create Index File
- 2. Create Index File
- 9. End Create Index File

Choice
2

LINK FIELDS 1.03 1 1

Enter a 2 as shown on the screen above. If you enter a 1, several screens will describe the program. The screens on the following pages will show all of your responses in green. Enter each response exactly as shown. Remember to press the New Line key to get from one input field to the next.

INTERNAL files - key-indexed access

Creating an index file (continued)

```

Create Index File                                     04-052
Master Filename ITEM                               Required
VOLID                                               Optional if master
Drive number 1                                       Optional if master
Key field length 5                                   Required
Position in record where key field starts 1         Required

Cmd 9 Cancel
? Help

LINK        FIELDS                                    1.03        1 1

```

The master file is ITEM. It is stored on the diskette in diskette drive 1 (or drive 3). The key is the item number. It starts in position one, and it is five characters long.

Remember to press the New Line key to get from one input field to the next. Press the Enter key after you complete each of these screens.

Create Index File		04-062
Index Filename <u>ITEMX</u>	Required	
VOLID _____	Optional when drive number provided and no index file exists. Not required if index file exists.	
Drive number <u>1</u>	Optional when VOLID provided and no index file exists. Not required if index file exists.	
Include duplicate keys in index file Y-Allow duplicate keys N-No duplicate keys <u>Y</u>		
LINK	FIELDS	1 1
		1.03
		Cmd 9 Cancel ? Help

The index file you are creating is ITEMX. This entry can be any filename. If this file does not already exist, your System/23 will create a new file. If this file does exist, your new index will replace any data stored in ITEMX. You are creating your index file on the same diskette as the master data file (diskette in drive 1 or 3). The Y response tells your System/23 that you may use the same item number for more than one item. If you do, you will have duplicate keys.

INTERNAL files - key-indexed access

Creating an index file (continued)

This screen may not always appear as you run INDEX.

```

Create Index File                                     04-082

This Customer Support Function may require
diskette work space. It will look for enough
space to create a work file called
INDEX.WORK. At the completion of this
function, the work file will be freed.

Work file VOLID                                     Optional when
  _____ drive number provided.

Work file drive number                               Optional when
  1                                                VOLID provided.

Cmd 9 Cancel
? Help

LINK          FIELDS                               1.03          1 1
```

This *work file* is a file that is created and used within the INDEX program. You will never be concerned with the contents of this file.

You are telling your System/23 to use the diskette in diskette drive 1 (or 3) for this file.

```

Create Index File 04-092

Information and messages to:
1-Screen only
2-Screen and printer
3-Printer only
2

Printer number Required when
10-First printer printer used
11-Second printer
10

List duplicate keys
Y-Display and/or print duplicate keys
N-Do not list duplicate keys
Y

Cmd 9 Cancel
? Help

LINK FIELDS 1.03 1 1

```

You enter 2 to tell your System/23 to display and print any program messages. If you do not have a printer attached to your System/23, enter 1 for display only. The messages include any duplicate keys in the master file and any error messages.

The 10 refers to the device address of the system printer.

INTERNAL files - key-indexed access

Creating an index file (continued)

When INDEX completes your index, the following screen is displayed:

```

Create Index File                                     04-112

Statistics:
Master records read           4
Keys written                  4
Duplicate key strings         0
Duplicate keys                 0

Press Enter to continue                               Cmd 9 Cancel

LINK      FIELDS                                     1.03      1 1
```

This screen tells you how many records are in the file ITEM. It also tells you how many keys are in the index and if there are any duplicate keys.

To continue, press the Enter key.

Create Index File 04-142

Create Index File successful.

Press Enter to return to menu

LINK FIELDS 1.03 1 1

00001	1
00002	2
10008	4
12345	3
↑	↑
Key in	Record
ITEM	number

This is what is in the index. Now press the Enter key.

INTERNAL files - key-indexed access

Creating an index file (continued)

Create Index File Menu

04-010

Choose one of the following:

1. Description of Create Index File
2. Create Index File

9. End Create Index File

Choice

9

LINK

FIELDS

1.03

1 1

You enter 9 to end the INDEX program.

Reading a record from a key-indexed file

Now that you have created your index, you are ready to read a record from your ITEM file. Enter the following:

```
CLEAR
10 REM READ FROM FILE WITH INDEX
20 DIM D$*20
30 OPEN #1:"NAME=ITEM,KFNAME=ITEMX",INTERNAL,OUTIN,KEYED
40 READ #1,USING 50,KEY="12345":I$,D$,Q,P,P1
50 FORM C 5,C 20,N 6,N 9.2,N 17.2
60 PRINT USING 70:I$,D$,Q,P,P1
70 FORM C 6,C 20,PIC(####),X 4,PIC($$$.#),X 4,PIC($$$$#.#)
80 END
```

What does this program do? It opens a key-indexed file and reads the record whose key is 12345. The contents of the record are displayed on the screen.

Let's look at the OPEN statement in line 30. This line has two things you haven't seen before.

```
30 OPEN #1:"NAME=ITEM,KFNAME=ITEMX",INTERNAL,OUTIN,KEYED
```

KFNAME is key filename. This is the name of the index file that you created.

KEYED means key-indexed. This tells your System/23 that the records in the ITEM file will be accessed by way of a key index.

INTERNAL files - key-indexed access

Reading a record from a key-indexed file (continued)

Now let's look at the READ statement in line 40.

```
40 READ #1, USING 50, KEY="12345": I$, D$, Q, P, P1
```

This statement is similar to READ for a relative record I/O file. But, instead of entering REC=3, you enter KEY="12345".

Now run the program:

```
RUN
12345 SCREWS      0850    $ .11    $93.50
```

Notice what was displayed.

The record with 12345 in the key field (first five positions) was the only record read from ITEM.

Writing and deleting records in a key-indexed file

Now you're ready to add a record to your key-indexed file. Since this is a record I/O file, you need to use the WRITE statement. Your index file is automatically updated when you update your master data file using a key-indexed file.

Let's see how this works. Enter the following:

```
10 REM ADD RECORD TO KEY-INDEXED FILE
40 PRINT "ENTER ITEM NO.,DESCRIPTION,QUANTITY,COST"
45 INPUT I$,D$,Q,P
60 WRITE #1,USING 50:I$,D$,Q,P,Q*P
DEL 70
```

List the new version of your program:

LIST

```
00010 REM ADD RECORD TO KEY-INDEXED FILE
00020 DIM D$*20
00030 OPEN #1:"NAME=ITEM,KFNAME=ITEMX",INTERNAL,OUTIN,KEYED
00040 PRINT "ENTER ITEM NO.,DESCRIPTION,QUANTITY,COST"
00045 INPUT I$,D$,Q,P
00050 FORM C 5,C 20,N 6,N 9.2,N 17.2
00060 WRITE #1,USING 50:I$,D$,Q,P,Q*P
00080 END
```

Now run this program with the following responses:

```
RUN
ENTER ITEM NO.,DESCRIPTION,QUANTITY,COST
?H1000,HAMMERS,7,10.98
```

Item number H1000 is added to the end of the ITEM file. The index is automatically updated to show that key H1000 is in record number 5.

INTERNAL files - key-indexed access

Writing and deleting records in a key-indexed file (continued)

New keys are added to the end of the index file. But the computer will read them in logical order. To make your programs run faster, we recommend that you run the INDEX Customer Support Function after you change a key-indexed file.

Now let's see what happens when you update a record in a key-indexed file. Enter the following:

```
10 REM UPDATE RECORD IN KEY-INDEXED FILE
40 READ #1,USING 55,KEY="00002":Q,P
45 Q=Q-100
55 FORM X 25,N 6,N 9.2
60 REWRITE #1,USING 65,KEY="00002":Q,Q*P
65 FORM X 25,N 6,X 9,N 17.2
```

List your program:

LIST

```
00010 REM UPDATE RECORD IN KEY-INDEXED FILE
00020 DIM D$*20
00030 OPEN #1:"NAME=ITEM,KFNAME=ITEMX",INTERNAL,OUTIN,KEYED
00040 READ #1,USING 55,KEY="00002":Q,P
00045 LET Q=Q-100
00050 FORM C 5,C 20,N 6,N 9.2,N 17.2
00055 FORM X 25,N 6,N 9.2
00060 REWRITE #1,USING 65,KEY="00002":Q,Q*P
00065 FORM X 25,N 6,X 9,N 17.2
00080 END
```

This program searches the index for the key "00002." This key is in record 2 of ITEM, so record 2 is read.

The quantity and total value are updated, and the record is rewritten. Notice in line 60 that you specify the key, and not the relative record number, of the record to be rewritten.

```
60 REWRITE #1,USING 50,KEY="00002":Q,Q*P
```

Writing and deleting records in a key-indexed file (continued)

Run your program:

```

RUN

```

```

RUN

```

Note: The only field that can't be changed on a rewrite is the key field.

You can use an index file to indicate a deleted record. Enter the following:

```

DEL 45
60 DELETE #1,KEY="00002":

```

```

DEL 45
60 DELETE #1,KEY="00002":
RUN

```

```

RUN

```

The index file is coded so that item "00002" in the ITEM file cannot be accessed.

Let's enter one more program to make sure that all of our changes have been made to the ITEM file. Enter:

```

CLEAR
10 REM DISPLAY THE RECORDS IN THE FILE
20 DIM D$*20
30 OPEN #1:"NAME=ITEM,KFNAME=ITEMX",INTERNAL,OUTIN,KEYED
40 PRINT "ENTER ITEM NUMBER TO BE CHECKED"
50 INPUT A$
60 IF A$="LAST" THEN STOP
70 READ #1,USING 80,KEY=A$:I$,D$,Q,P,P1
80 FORM C 5,C 20,N 6,N 9.2,N 17.2
90 FORM C 6,C 20,PIC (####),X 4,PIC ($$$$.##),X 4,PIC ($$$$#.##)
100 PRINT USING 90:I$,D$,Q,P,P1
110 GOTO 40
120 END

```

INTERNAL files - key-indexed access

Writing and deleting records in a key-indexed file (continued)

Now run the program, and check item numbers 00001, 10008, H1000, and 00002:

RUN

```
RUN
ENTER ITEM NUMBER TO BE CHECKED

?00001
00001 NUTS          1800    $ .29    $522.00
ENTER ITEM NUMBER TO BE CHECKED

?10008
10008 NAILS         5000    .02     $100.00
ENTER ITEM NUMBER TO BE CHECKED

?H1000
H1000 HAMMERS       0007    $10.98   $76.86
ENTER ITEM NUMBER TO BE CHECKED

?00002
-

RUN                ERROR 99    ITEMX/VOLID/1  4272                70
```

The error 4272 in line 70 proves that item 00002 was deleted. Look in your *Messages* manual for error code 4272. Then, press the Error Reset key.

When the asterisks appear on the screen, enter:

GO END

Chapter summary

A key-indexed file is a record I/O file with an associated index file. The index file cannot be used by itself; it must be used with a master file.

The index file is created by way of a Customer Support Function. You enter LINK INDEX to create the index.

The index includes a key, or set of characters, used to identify each record and the associated relative record numbers.

Statements used to process a key-indexed file include:

Statement	Programming concerns
OPEN	KFNAME=key filename access method=KEYED
READ	KEY="identifying characters"
WRITE	do not specify REC= or KEY=
REWRITE	KEY="identifying characters"
DELETE	KEY="identifying characters"

You can create more than one index for the same master file. For example, with a master file of names and addresses, you could have two different index files:

- One with zip codes used as the key.
- One with names used as the key.

INTERNAL files - key-indexed access

Exercises

Question 1

What command must you enter to create an index file for a master data file?

Answer: _____

Question 2

Match the parts of the following OPEN statement with their corresponding functions.

80 OPEN #3:"NAME=ABC,KFNAME=ABCX",INTERNAL,OUTIN,KEYED

- | | |
|---------|--------------------------------------|
| __ABC | a. The name of the index file. |
| __ABCX | b. The file will be accessed by key. |
| __#3 | c. The name of the master data file. |
| __KEYED | d. The file reference number. |

Question 3

Assume that the following key-indexed file EMP is stored on the diskette in drive 1. Its index file is EMPX. The first six characters in each record are the keys.

573277	JOHN DOE	06-06-38	06-01-66
120089	MARY SMITH	09-13-52	11-15-76
007719	JANE GREEN	01-04-52	07-16-79

What would be displayed if you ran the following program?

```
10 OPEN #1:"NAME=EMP//1,KFNAME=EMPX",INTERNAL,INPUT,KEYED
20 READ #1,USING 30,KEY="120089":E1$,N1$
30 FORM C 6,X,C 10
40 READ #1,USING 30,KEY="007719":E2$,N2$
50 PRINT USING 30:E1$,N1$
60 PRINT USING 30:E2$,N2$
70 END
```

Answer: _____

INTERNAL files - key-indexed access

Answers

Question 1

LINK INDEX

Question 2

c ABC
a ABCX
d #3
b KEYED

Question 3

120089	MARY SMITH
007719	JANE GREEN

Chapter 6. Diskettes and diskette drives

Introduction

In this chapter, you will learn how to use the dual-station System/23 and how to share files on one or more stations. If you do not have a second computer attached to your System/23, or if you do not want to share files between programs, you may want to skip this chapter and go on to Book VII. (You can refer to Chapter 6, "Commands," in your *Operator Reference* manual for information on the system commands discussed here.)

In this chapter, you will learn how to specify the share state of a file. You will learn how to allow a file to be used by one or more programs on one or more work stations. (You can refer to "Device Sharing," in your *Basic Language Reference* manual for more information.)

Objectives

Upon completion of this chapter, you should be able to do the following:

- Specify the share state and span of control by using the OPEN statement.
- Identify the share state of a file by using the DIR command.
- Close a file by using the PROTECT command or CLOSE statement.
- Identify which computer is using the diskette unit by reading the status line.

If you are familiar with these tasks, try the exercises at the end of this chapter. If not, read through the chapter before going on to the exercises.

Diskettes and diskette drives

The share state

Your diskette unit permits two computers to access the same data. You can also open a file more than once in the same program. For that reason, you will need to know how to use the different *share states*.

A *share state* tells your System/23 whether or not your file can be *opened* more than once, and if so, how. We use the word *open* here as an attempt to gain control of a file. The three share states are:

- SHR—Both OPENs can open the file and read data from it, but only one OPEN can write data into it.
- SHRI—The other OPEN can open the file and read data from it, but cannot write data into the file.
- NOSHR—The second OPEN will cause an error until the first OPEN is closed.

In these share states, the *first open* refers to the first OPEN statement that tries to access the file. The *second OPEN* refers to the second OPEN statement that tries to access the file when it is already opened.

The share states also depend upon the open mode:

- INPUT—read only mode
- OUTPUT—write only mode
- OUTIN—read and write mode (update)

One computer cannot use the files stored on another computer. But, both computers can use a file stored on the shared diskette unit.

Let's look at an example. Assume that you have a dual-station System/23. One computer is called station "01" and the other is called station "02".

Now, assume that the person using station 01 is running the following program.

```
10 OPEN #1:"NAME=ABC//3,SHR",INTERNAL,OUTPUT
20 FOR X=1 TO 100
30 WRITE #1,USING 50:X
40 NEXT X
50 FORM N 4
60 END
```

Because of the SHR in line 10, a program on station 02 can open and read from the file ABC. However, only one program can have a file open for output at one time, so the program on station 02 cannot write to the file. The program on station 01 would have similar restrictions if this program were running on station 02.

Diskettes and diskette drives

Device sharing information

Sometimes you may have to wait for the diskette unit if the unit is being used by the other computer. The amount of time you have to wait depends on what the other computer is doing.

How can you tell if the other computer is using the diskette unit? Look at the status line on your screen, at the +1 shown in the picture.



This field is blank when you are not using the diskette unit and are not trying to access it. However, whenever you try to access the unit, either a +1 or a +0 will be displayed in this field.

If a +1 is displayed, your request to use the diskette unit was successful. The +1 is displayed until you stop using the unit.

If a +0 is displayed, the other computer is already using the diskette unit. You cannot use it at this time. You must wait for the other computer to finish using the unit before you can access it.

You can do one of two things while the other computer is using the diskette unit. You can:

- Wait for the diskette unit to be free.

You may check with the other operator and find that there will be only a short wait until the diskette unit becomes available. So you decide to wait. When the unit becomes available, the +0 on the status line will change to +1.

- Cancel the job.

You may decide to cancel your job instead of waiting for the other computer to finish. If so, do the following: Hold down the Cmd key and press the Attn key. An error 6010 will appear with action code 20. Then you should hold down the Cmd key and press the Error Reset key.

When the asterisks appear on the screen, enter GO END, and then enter CLEAR ALL. Then you are ready to start another job.

CLEAR ALL will cause all data to be lost. It's almost like turning the machine off, but you don't have to set the time and date again.

Diskettes and diskette drives

File sharing information

Let's look at another example. This time, the person using station 01 is running the following program.

```
10 OPEN #1:"NAME=ABC//3,SHRI",INTERNAL,INPUT
20 FOR X=1 TO 100
30 READ #1:X
40 NEXT X
50 END
```

The file ABC is opened for input, with SHRI. That means that the person using station 02 could be reading the file ABC, also.

For example, the person using station 02 could be running the following program.

```
20 OPEN #7:"NAME=ABC//3,SHRI",INTERNAL,INPUT
40 READ #7:NUMBER
60 IF NUMBER=100 THEN 40 ELSE PRINT "END OF FILE"
80 END
```

Both programs would be reading data from the same file on the diskette unit.

However, line 20 of the second program could not look like this:

```
20 OPEN #7:"NAME=ABC//3,SHR",INTERNAL,OUTIN
```

Nor could it look like this:

```
20 OPEN #7:"NAME=ABC//3,NOSHR",INTERNAL,INPUT
```

The first program specified sharing for *input only*. NOSHR specifies no sharing.

Let's look at one more example. Suppose the person using station 02 is running the following program.

```
10 OPEN #5:"NAME=ABC//3,NOSHR,RESERVE",INTERNAL,OUTPUT
20 WRITE #5,USING 30:200,300,400,500
30 FORM N 6.2,N 6,PIC(#####),PIC(ZZZ###)
40 FOR X=1 TO 66
50 PRINT #255:
60 NEXT X
70 FOR X=1 TO 66
80 PRINT #255:X
90 NEXT X
100 END
```

The RESERVE in line 10 specifies that the OPEN,NOSHR share state is to be permanently associated with this file and station. Until a CLOSE #5,RELEASE: is issued, no OPENS are permitted on the other computer.

Without the word RESERVE, nobody else can use the file ABC at the same time as the person running the program. With the word RESERVE, the file is unavailable to station 01 even after station 02 closes the file, unless RELEASE is used.

You can reset the RESERVE by using the CLOSE statement, like this:

```
35 CLOSE #5,RELEASE:
```

Diskettes and diskette drives

The DIR and PROTECT commands

In *Learning to Use System/23*, you learned how to use the DIR command to list and describe the contents of a diskette drive. If you are not familiar with DIR, refer to Chapter 6, "Commands," in your *Operator Reference* manual.

We must point out two things about using DIR with a dual-station System/23:

- If the person using station 02 is running a program that uses file ABC on drive 3, then the person using station 01 is locked out of the diskette unit while station 02 reads or writes to that file.
- If a file is left open when a diskette is removed from a drive, the share state will appear in the DIR listing.

If a file is accidentally left open, you can use the PROTECT command to close it.

Let's look at an example.

Assume that the person on station 01 accidentally removed the diskette from drive 3 before the file ABC was closed. (This situation could also occur during a loss of power.)

Then the person on station 01 reinserted the diskette and entered DIR 3. The DIR listing would indicate that the file ABC was open. It wasn't closed before the diskette was removed.

Until the person on station 01 closes the file ABC, the person on station 02 cannot use file ABC. Therefore, the person on station 01 should enter:

```
PROTECT ABC//3,CLOSE
```

Then the file would be closed and could now be reopened.

Remember: Any time a file is accidentally left open, you can close the file by reinserting the diskette and entering the PROTECT command.

If the file left open was being used for output, however, records may have been lost. Use PROTECT CLOSE *only* if you are sure no other station or program is using the file.

You should not issue a PROTECT CLOSE or a PROTECT RELEASE command on a file that is being used by either station. Unpredictable results may occur.

Diskettes and diskette drives

Other BASIC commands

Remember from *Learning to Use System/23* that you use the RENAME command to change the name of an existing file. You cannot use the RENAME command for a file that is already opened.

While you are in the process of renaming a file on the diskette unit, the file must not already be open.

You cannot use the DROP and FREE commands to get rid of a file that is already opened. You must first close the file.

The LINK and LOAD commands, as you know, place a copy of a program, a display file, or a Customer Support Function in your work area. These commands will always open the programs and files for INPUT SHRI. This means that two people can load one program at the same time.

The SAVE and REPLACE commands open a file for OUTPUT NOSHR. This means that only one person can save or replace any particular program or display file at a time.

You should not use the VOLID command to change the diskette VOLID if there are open files on that diskette. If you do, you may not be able to continue processing the files.

For more information on the dual-station System/23, refer to "Device sharing" and "File sharing" in the *BASIC Language Reference* manual.

Chapter summary

The dual-station System/23 allows two people to use your system at one time. Both stations have access to the diskette unit.

The files on the diskette unit can be shared by both computers. However, certain share states apply:

- SHR—Two OPENs can open the file and read data from it, but only one OPEN can write data into it.
- SHRI—The second OPEN, at the same time, can open the file and read data from it, but cannot write data into the file.
- NOSHR—The second OPEN cannot open the file until the first OPEN is closed.

You specify the share state in the OPEN statement like this:

```
OPEN #1: "NAME=XY//3, SHR", INTERNAL, OUTPUT
```

You can close an open file in a program by using the CLOSE statement or by normal termination of the program. If an error situation exists and the file remains open, you can close an open file on a diskette by using the PROTECT command. Use the PROTECT command only if you are sure no other station or program is using the file.

You cannot RENAME, DROP, FREE, SAVE, or REPLACE a file while it is opened. Both stations can run the same program at one time by using LINK or LOAD.

Diskettes and diskette drives

Exercises

Question 1

Assume the person using station 01 is running the following program.

```
10 OPEN #8:"NAME=XYZ//3,SIZE=0,RECL=24,SHR",INTERNAL,OUTPUT
20 FOR NUMBER=1 TO 80
30 WRITE #8,USING 50:NUMBER
40 NEXT NUMBER
50 FORM N 24.2
60 RESTORE #8:
70 FOR X=1 TO 20
80 READ #8,USING 50:X
90 PRINT X
100 NEXT X
110 END
```

Which of the following OPEN statements could be executed on station 02?

- a. OPEN #2:"NAME=XYZ//3,SHR",INTERNAL,INPUT
- b. OPEN #12:"NAME=ABC//3,NOSHR",INTERNAL,OUTPUT
- c. OPEN #8:"NAME=XYZ//1,SHR",INTERNAL,OUTPUT
- d. OPEN #8:"NAME=XYZ//3,SHR",INTERNAL,OUTPUT

Question 2

What command do you use to close a file that was accidentally left open?

Answer: _____

Answers

Question 1

- a. Yes—XYZ was originally opened SHR
- b. Yes—ABC is a different file
- c. Yes—Drive 1 is not on the diskette unit
- d. No—Only one station can output to a file

Question 2

PROTECT

Question 1

- a. Yes—XYZ was originally named SHI.
- b. Yes—ABC is a different firm.
- c. Yes—Dave is not on the list.
- d. No—Only one answer is correct.

Question 2

10/10/10

READER'S COMMENT FORM

SA34-0126-1

VI. Data Files and Diskettes

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or the IBM branch office serving your locality.

Corrections or clarifications needed:

Page Comment

Cut or Fold Along Line

Please indicate your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Reader's Comment Form

READER'S COMMENT FORM

SA34-0126-1

V. Dan Filer and D. ...

Your comments assist us in improving the usefulness of our publications. They are an important part of the input that is being used to improve our products. IBM may not and does not intend to be held liable for any errors or omissions that may appear in this publication. IBM is not responsible for any damage or loss of data that may result from the use of the information contained in this publication.

Please do not use this form for advertising or promotional purposes. The information on this form is for editorial use only.

Cut Along Line

Fold and tape

Please Do Not Staple

Fold and tape



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK
POSTAGE WILL BE PAID BY ADDRESSEE



IBM Corporation
Information Development, Dept 27T
P.O. Box 1328
Boca Raton, Florida 33432

Fold and tape

Please Do Not Staple

Fold and tape



Please do not use this form for advertising or promotional purposes. The information on this form is for editorial use only.





SA34-0126-1
Printed in U.S.A.