HEWLETT PACKARD

MIKSAM ROM
Owner's Manual

HP-86/87

December 1982

Reorder Number
00087-90614

Printed in U.S.A.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Contents

Getting Started

Introduction

The MIKSAM ROM is used for creating and maintaining key files. MIKSAM stands for *Multiple Indexed Keyed Sequential Access Method.* This means several key files can be used for supporting one or more direct access files. MIKSAM allows up to 12 key files to be open simultaneously. The number of data files accessed with the keys depends on the specific application.

To use the MIKSAM ROM properly, several HP products are needed. A minimal configuration includes these items:

- HP 82936A ROM Drawer.
- One Disc Drive.
- HP-86 or HP-87 Personal Computer.

Depending on the program files, key files, and data files associated with your application, one disc drive may not provide sufficient mass storage capacity. Additional disc drives can be added when necessary. The MIKSAM ROM uses 4K bytes of user memory (RAM) as a buffer. The remaining user memory is available for application programs.

ROM Installation

The MIKSAM ROM can be inserted into any vacant slot on the ROM drawer. You can remove the protective cap over the slot by placing the eraser end of a pencil through the circular hole underneath the slot and pressing upwards.

Next, position the MIKSAM ROM so that the connector pins face down and the beveled edge is towards the plug-in side of the drawer. Be careful not to touch the connector pins. Gently press the MIKSAM ROM into the slot until it is flush with the drawer surface.

> **Note:** Additional information is provided in appendix A of this manual, on the *HP 82936A ROM Drawer Instruction Sheet,* and in section 2 of the introductory manual supplied with your computer.

Once the MIKSAM ROM has been installed and the ROM drawer is plugged into a module port, your HP-86/87 can be switched on. To verify that the MIKSAM ROM is properly installed, press the keys (M) (I)(K)(S)(A)(M)(END LINE). The following message should be displayed:

```
(c) 1982 Peachtree Software Inc., Atlanta, Ga.
```

The MIKSAM ROM requires 4K bytes of user memory (RAM) to operate properly. These instructions allow you to determine the amount of user memory remaining:

a. Execute the SCRATCH command.

b. Press (LIST).

c. The number of bytes available in user memory is displayed. This amount of user memory is available for application programs. The 4K bytes used by the ROM are already subtracted from the number displayed because memory for the ROM is taken when power is applied.

CAUTION

Do not insert or remove the ROM drawer containing the MIKSAM ROM when your HP-86/87 is switched on. Inserting or removing drawers can result in serious damage to the MIKSAM ROM, other ROMs, and the internal circuitry of your computer. Always check the POWER light to verify that your HP-86/87 is switched **off** before inserting or removing a drawer. The computer should also be plugged into a grounded electrical outlet.

Definitions

Here are some terms you will encounter in this manual:

ASCII collating sequence	The relative ordering of decimal codes assigned to all characters and keys on the HP-86/87, as defined by the *American Standard Code for Information Interchange.*
corrupt	A disc file condition caused by interruption of disc output operations. In a **corrupt** file, data is incomplete or invalid because the extent of a specific disc write is uncertain. A power outage can cause this situation, for example.
data file	A direct access disc file where specific records can be accessed with record numbers. The buffer for these files is set up with BASIC ASSIGN# statements.
data type	The internal representation used for evaluating variables, constants, or expressions. There are two data types with MIKSAM statements, *numeric* and *string*.
header file	A separate disc file that contains system information for accessing key files or data files.
input parameter	A constant, variable, or expression used to pass necessary data to a MIKSAM statement.
key file	A disc file used to quickly access record numbers given a key or retrieve the key-record number pairs in order. These files are created with MIKSAM MAKE_KEY_FILE statements.
key-record number pair	The unit of information stored in the key file. The key is a string value searched for in the key file. A record number is associated with each key for accessing the data file.
numeric data type	A *numeric* variable, constant, or expression that can be used for passing numeric data in MIKSAM statements.
output parameter	A variable to which specific values can be returned from MIKSAM statements. After the statement has been executed, this value can be tested in the program.
parameter	Constants or variables listed after MIKSAM statements that enable the exchange of values between program statements and the MIKSAM ROM during program execution. There are three types of parameters—*input*, *output*, and *update*. A listing is provided on page 21.
pointer	A numeric value in one cell of a data structure that contains the address of another cell and links different cells in the data structure together.

record	Information accessed in the data file with direct access BASIC READ# and PRINT# statements. All records in a given file have the same fields although each field can contain different information.
statement	Any instruction that can be used in an executable program. MIKSAM statements can be executed on your HP-86/87 if the MIKSAM ROM has been properly installed.
string data type	A variable, constant, or expression evaluating to a specific sequence of ASCII characters.
update parameter	A variable used for passing data to and from a MIKSAM statement. The updating of the variable depends on the successful execution of the statement. An *update* parameter has the properties of both *input* and *output* parameters.

File Operations

The following table summarizes the capabilities of MIKSAM statements for maintaining key files and presents typical data file operations.

Statements	Key Files	Data Files
MAKE_KEY_FILE	Creating key files.	Initializing and expansion of files.
KILL_KEY_FILE	Purging key files.	—
OPEN_KEY_FILE	Opening key files.	—
CLOSE_KEY_FILE	Closing key files.	—
CREATE_KEY	Adding keys.	Adding records.
DELETE_KEY	Deleting keys.	Deleting records.
SEEK_FIRST	Find the first key.	Retrieve first record.
SEEK_END	Find the last key.	Retrieve last record.
SEEK_NEXT_KEY	Find the next key.	Retrieve records in key order.
SEEK_PRIOR_KEY	Find the previous key.	Retrieve records in reverse key order.
SEEK_KEY	Find the record number of a key.	Accessing specific records.
SET_UP	Setting up buffers.	—
M_STATUS	Managing file space.	—

Programmer Responsibilities

This manual is intended for application programmers. Knowledge of data base programming techniques is assumed. Familiarity with indexed keyed sequential access methods is desirable. The B-tree is the data structure used in all MIKSAM key files. It is the responsibility of the programmer to maintain direct access files using BASIC statements. Details about these statements are provided in the *HP-86/87 Operating and BASIC Programming Manual.* The following materials provide a background in data base programming and file structures:

C. J. Date, *An Introduction to Database Systems,* Second edition, Addison-Wesley, Reading, Massachusetts, 1977, 536 pages.

Gio Wiederhold, *Database Design,* McGraw-Hill, New York, 1977, 658 pages.

Ellis Horowitz and Sartaj Sahni, *Fundamentals of Data Structures,* Computer Science Press, Potomac, Maryland, 1976, 564 pages.

Donald E. Knuth, *The Art of Computer Programming: Volume 3: Searching and Sorting,* Second edition, Addison-Wesley, Reading, Massachusetts, 1973, 722 pages.

Additional information about MIKSAM key file structure and performance is presented in section 4.

Section 2

MIKSAM Statements

Introduction

MIKSAM statements can be executed in either program or calculator mode on the HP-86/87. All MIKSAM statements consist of a **keyword** followed by one or more **parameters.** The parameter list can contain constants, variables, and expressions and is used for exchanging data with the routines invoked by MIKSAM statements.

Three types of parameters are used—*input*, *output*, and *update*. *Input* parameters are used for supplying information to specific routines. *Output* parameters are assigned new values and return information to application programs. *Update* parameters supply information to statements and can also return values to a program. Here are some guidelines:

Input parameters —expressions, constants, or variables can be used. Depending on the parameter, use either a *numeric* or *string* data type.

Output parameters —use variables in all cases. This is because values for *output* parameters are returned after statement execution. Such values cannot be assigned to constants or expressions.

Update parameters —use variables with previously assigned values. Actual updating depends on the outcome of statement execution.

All variables or expressions used for parameters must result in a defined value when evaluated. The application program must initialize or set *input* and *update* parameters. Values returned to *output* and *update* parameters must also be checked. Refer to page 21 for a comprehensive listing of parameters.

In the remainder of this section, specific information is presented about each of the 13 MIKSAM statements. Each statement description includes the parameter list, sample statements, and the status codes that can be returned. This format is used for describing all parameters:

Parameter (parameter type, data type)

The *parameter type*, described earlier, can be *input*, *output*, or *update*. The *data type* can be *numeric* or *string*. For statements to be executed properly, parameters must be supplied in the order indicated. Parameters used with the SEEK_KEY statement, for example, are listed as follows:

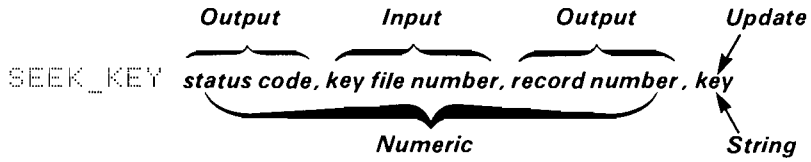
a. ***Status code (output, numeric)***

b. ***Key file number (input, numeric)***

c. ***Record number (output, numeric)***

d. ***Key (update, string)***

The keyword for this statement, SEEK_KEY, is always followed by the parameter list:



Correct SEEK_KEY SC,1,RECNUM,KEY$

Incorrect SEEK_KEY SC,FILENUM,RECNUM,KEY$+V$

Reason Expressions should not be used as *update* parameters since data can be returned to them.

To access file records directly, it is necessary to supply a record number. This number corresponds to the sequential ordering of the data file. Key files enable you to look up the correct record number in a data file without checking any adjacent records.

With the SEEK_KEY statement, you could supply a customer's last name and obtain a record number to access a corresponding record in a data file. The record number can be used with both READ# and PRINT# statements. For additional information about the SEEK_KEY statement, refer to page 18.

Note: An abbreviated format can be used for entering MIKSAM statements on your HP-86/87. All characters in the keyword except the starting character and characters preceded by an underscore can be omitted. For example, the SEEK_KEY statement can be entered as S_K. Another example is entering the MAKE_KEY_FILE statement as M_K_F. The entire keyword is shown when programs are listed. The M_STATUS statement cannot be abbreviated.

The MAKE_KEY_FILE Statement

The MAKE_KEY_FILE statement creates a key file on disc. The key file is pre-allocated and consists entirely of record pointers and keys. Since the file is stored in the first vacant space on disc, it is recommended that only packed or newly initialized discs be used for storing key files.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates if key file creation is successful.

b. ***Key file specifier (input, string)*** Provides the *file name* and *mass storage unit specifier (msus)* for creating the file. No other files on the volume specified should have the same name. When the *msus* is omitted, the key file is stored on the default mass storage device.

c. ***Key length (input, numeric)*** Specifies the number of bytes in the key. Values 1 through 60 can be used. Strings of this length then need to be used for the *key* parameter.

d. ***Key file extent (input, numeric)*** Specifies the number of sectors for the key file. This amount of disc space needs to be on-line. A minimum of 13 sectors is needed. The maximum extent is the number of sectors that can be stored on one volume with your system. Note that additional sectors are occupied by the key file when null disc spaces are used. Only the specified extent can be accessed.

Sample Statements:
```
MAKE_KEY_FILE SC,"CUSTOMER:D701",15,30
M_K_F STATUS,"CUSTKEY:D722",60,NUM_REC
```

The number of key-record number pairs created in your key file determines the number of records that can be indexed by the key file. With the MAKE_KEY_FILE statement, this is specified in sectors. Refer to appendix B for converting the number of key-record number pairs and key length to key file extent.

Status Codes

Values	Conditions	Notes
120	Device address is not correct.	Check *msus* and device address.
122	Invalid key length.	Must be in range 1 through 60 bytes.
125	Invalid extent.	Must be at least 13 sectors.
130	Not enough room for file.	Purge unneeded files.
132	File already exists.	Must change the *file specifier* or delete the existing file.
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful file creation.	—

The KILL_KEY_FILE Statement

This statement deletes key files. Only key files created with the MAKE_KEY_FILE statement can be deleted with the KILL_KEY_FILE statement. After deleting, the disc space becomes available for other files.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates successful deletion and abnormal conditions.

b. ***Key file specifier (input, string)*** Identifies the file to be deleted. If the *msus* is not included, only the default mass storage location is checked.

Sample Statements:
```
KILL_KEY_FILE STATUS,"CUSTOMER:D701"
K_K_F RET_CODE,KFILE$
```

Only files identified as type PKEY in a directory listing can be deleted with the KILL_KEY_FILE statement. The MIKSAM ROM must already be installed when the CATALOG command is entered.

Status Codes

Values	Conditions	Notes
120	Incorrect device address.	Check *msus* specified.
126	File not PKEY type.	Obtain directory of disc with ROM installed.
133	File in parameter is not on-line.	Check *msus*.
134	File is open.	Execute CLOSE_KEY_FILE statement, then delete.
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful deletion.	—

The OPEN_KEY_FILE Statement

This statement opens specific key files and assigns numbers to them. The number is used for accessing the file with other MIKSAM statements and must be used until the file is closed. A pointer is positioned at the beginning of each key file when it is opened. Up to 12 key files can be open simultaneously.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates if the file is properly opened.

b. ***Key file specifier (input, string)*** Locates the disc file to be accessed. The *msus* can be included.

c. ***Key file number (input, numeric)*** Assigned to the file when it is opened. The numbers 1 through 12 can be used. Do not use numbers currently assigned to other key files.

Sample Statements:
```
OPEN_KEY_FILE STAT,"CUSTOMER",2
O_K_F SC,"CUSTKEY:D722",FILE
```

If you attempt to open a file with the same name as one that is already open, a status code is set. Each file that is opened must have a different name.

Status Codes

Values	Conditions	Notes
103	File opened—it was not properly closed.	Check for corrupt data.
120	Peripheral address not correct.	Check *msus* specified.
121	Invalid file number.	File number is currently in use or is not in range 1 through 12.
126	File is not a key file.	Must be type PKEY.
133	File in parameter is not on-line.	Check *msus*.
134	File is open.	—
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful opening.	—

The CLOSE_KEY_FILE Statement

The CLOSE_KEY_FILE statement is used to close key files. If a specific key file is not closed, the next OPEN_KEY_FILE statement identifying this file will indicate that is was not closed properly. After a CLOSE_KEY_FILE statement has been successfully executed, the *key file number* of the closed file can be reused to open another key file.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates abnormal file closures.

b. ***Key file number (input, numeric)*** Identifies the file to be closed. Use the number assigned to the file by the respective OPEN_KEY_FILE statement.

Sample Statements:
```
CLOSE_KEY_FILE CODE,12
C_K_F SC,FILE
```

It is not necessary for the system to write out buffers to key files when they are closed because all output to key files is **direct**. This means that individual changes to key files are written out to disc immediately. This feature protects the key files from corruption.

Status Codes

Values	Conditions	Notes
121	Invalid key file number.	Must be in range 1-12.
135	File is closed.	—
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful closure.	—

The CREATE_KEY Statement

This statement is used for inserting key-record number pairs into key files. All keys are inserted according to the ASCII collating sequence. This means that numbers are inserted before letters and that uppercase letters precede lowercase letters. Duplicate keys are inserted after any keys with the same value. It is the programmer's responsibility to maintain the logical relationship between key files and data files.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates successful insertion and whether or not file space is becoming limited. Appropriate action needs to be taken if there is no space in the key file.

b. ***Key file number (input, numeric)*** Identifies the file in which the key is to be inserted. Use the number assigned with the OPEN_KEY_FILE statement in the same session.

c. ***Record number (input, numeric)*** Identifies specific records in data files when used with READ# and PRINT# statements.

d. ***Key (input, string)*** Provides the string value used for inserting key-record number pairs into key files.

Sample Statements:
```
CREATE_KEY STAT,2,REC_NUM,KEY$[1,15]
C_K SC,FILE,125,"JONES          "
```

If a SEEK_NEXT_KEY statement is executed after a CREATE_KEY statement, then the key following the key just inserted is returned. If a key is inserted after the last key, an end-of-file condition is returned.

Status Codes

Values	Conditions	Notes
102	Space low—less than 12 sectors remain.	Increase extent of the key file.
121	Invalid key file number.	Range 1 through 12 only.
122	Improper key length.	Check key length entered when key file created.
128	Invalid record number.	Evaluates to zero.
131	No space in key file.	Increase key file extent.
135	File is closed.	Use OPEN_KEY_FILE statement.
136	B-tree height is at maximum of seven.	Reduce key length or number of keys in file.
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful insertion.	—

The DELETE_KEY Statement

This statement removes key-record number pairs from key files. In order for deletion to be successful, the record number and key supplied must match a pair in the file specified. The *key* and *record number* parameters are updated to the next key-record number pair in the file when a deletion occurs. In most cases, disc space used by the deleted pair can be reused by the system.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates successful key deletion and error conditions.

b. ***Key file number (input, numeric)*** Identifies which key file a deletion occurs in. This should match the number assigned to the file with the OPEN_KEY_FILE statement.

c. ***Record number (update, numeric)*** Verifies the pair to be deleted. This must be the record number associated with the key. If end-of-file is encountered, this parameter is set to zero.

d. ***Key (update, string)*** Specifies the key to be deleted. The string length must match the *key length* parameter used when the file was created.

Sample Statements:
```
DELETE_KEY STAT,1,RN,CUSTKEY$
D_K RET_CODE,FILENUM,RECNUM,KEY$
```

When a SEEK_NEXT_KEY statement follows a DELETE_KEY statement, then the key originally appearing after the deleted key is returned. If the pair deleted was last in the key file, then the record number is set to zero and the value of the key parameter is not changed.

Status Codes

Values	Conditions	Notes
111	Key-record number pair not found.	If key is in file, check associated record number.
121	Invalid file number.	Must be in range 1 through 12.
122	Improper key length.	Check key length specified when file was created.
128	Invalid record number.	Evaluates to zero.
135	File is closed.	Use OPEN_KEY_FILE statement.
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful key deletion.	—

The SEEK_FIRST Statement

This statement places the pointer beside the first key-record number pair in the file specified. It should be used before each forward traversal of key files. The SEEK_FIRST statement is implicitly performed whenever a key file is opened.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates if the pointer is at start-of-file and detects abnormal conditions.

b. ***Key file number (input, numeric)*** Identifies the file. This parameter should be set to the value assigned with the OPEN_KEY_FILE statement during the current session.

Sample Statements:
```
SEEK_FIRST STAT,1
S_F SC,FILE
```

A SEEK_NEXT_KEY statement immediately following the SEEK_FIRST statement returns the first key-record number pair in a key file.

Status Codes

Values	Conditions	Notes
121	Invalid key file number.	Must be in range 1 through 12.
135	File is closed.	Use OPEN_KEY_FILE statement.
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful seek.	—

The SEEK_END Statement

Execution of this statement positions the pointer after the last key-record number pair in a key file. This statement is normally used before reverse traversals.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates if the pointer was positioned correctly and detects execution problems.

b. ***Key file number (input, numeric)*** Identifies the file.

Sample Statements:
```
SEEK_END STAT_CODE,3
S_E CODE,FILE
```

A subsequent SEEK_PRIOR_KEY statement initially returns the last key-record number pair in the specified file.

Status Codes

Values	Conditions	Notes
121	Invalid key file number.	Must be in range 1 through 12.
135	File is closed.	Use OPEN_KEY_FILE statement.
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful seek.	—

The SEEK_NEXT_KEY Statement

This statement finds the key-record number pair following the current pair. The first key-record number pair is returned when a SEEK_NEXT_KEY statement is executed after file opening. A series of these statements results in forward traversal of key files.

Order of Parameters:

a. ***Status code (output, numeric)*** Detects end-of-file and other conditions.

b. ***Key file number (input, numeric)*** Identifies the file to find the next key-record number pair in. Use the value assigned when the key file is opened.

c. ***Record number (output, numeric)*** Returns the number associated with the next key. This can be used for data file access.

d. ***Key (output, string)*** Returns the next key in a file according to the ASCII collating sequence.

Sample Statements:
```
SEEK_NEXT_KEY STAT,5,RECORD,KEY$
S_N_K STATUS,FILENUM,REC_NUM,KEY_VAL$
```

If the SEEK_NEXT_KEY statement is preceded by a SEEK_PRIOR_KEY statement, then an extra seek is required. This is because the direction of traversal is changing from descending to ascending. The extra seek is required to pass over the last key-record number pair retrieved.

Status Codes

Values	Conditions	Notes
101	End-of-file encountered.	Record number and key unchanged.
121	Invalid file number.	Must be in range 1 through 12.
122	Improper key length.	Must match key length specified during creation.
135	File is closed.	Use OPEN_KEY_FILE statement.
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful seek.	—

The SEEK_PRIOR_KEY Statement

This statement returns the key-record number pair preceding the current pair. When placed immediately after a SEEK_END statement, repeated execution of the SEEK_PRIOR_KEY statement results in reverse traversal of key files. A status code is set when start-of-file is encountered.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates successful retrieval and detects errors.

b. ***Key file number (input, numeric)*** Identifies what file the key-record number pair is retrieved from. Set this parameter to the value assigned to the file during opening.

c. ***Record number (output, numeric)*** Provides the number associated with the key. The record number can be used for accessing a data file.

d. ***Key (output, string)*** Returns the value of the preceding key.

Sample Statements:
```
SEEK_PRIOR_KEY SC,FI,RECN,KEY$
S_P_K CODE,6,REC,KEY_STRING$
```

Seeking prior keys usually changes the order in which keys are returned from ascending to descending. This is the case when the last seek was for the next key. Whenever such a change occurs, an extra seek is required to pass over the last key-record number pair returned. Also, when a SEEK_PRIOR_KEY statement follows a DELETE_KEY statement, the pointer is placed beside the key-record number pair originally preceding the deleted pair.

Status Codes

Values	Conditions	Notes
101	Pointer at start-of-file.	Record number and key unchanged.
121	Invalid file number.	Must be in range 1 through 12.
122	Improper key length.	Use key length file was created with.
135	File is closed.	Use OPEN_KEY_FILE statement.
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful seek.	—

The SEEK_KEY Statement

The SEEK_KEY statement searches for a key and returns the associated record number. This record number can be used to access data files. When a specific key is not found, the *key* and *record number* parameters are updated to the next pair in the file unless end-of-file is encountered.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates if the seek is successful.

b. ***Key file number (input, numeric)*** Identifies the file to be accessed. Set this parameter to the number assigned when the file is opened.

c. ***Record number (output, numeric)*** Returns the number associated with the key specified. This number can be used for accessing data files.

d. ***Key (update, string)*** Identifies the key to be retrieved from the key file.

Sample Statements:
```
SEEK_KEY STATUS_CODE,1,RECORDNUM,KEYVAL$
S_K RETCODE,FILE,RNUM,KEY$
```

When duplicate keys exist in a file, the record number of the first key-record number pair added to the file is returned. Repeated execution of the SEEK_KEY statement always returns this value. To obtain record numbers associated with remaining duplicate keys, use the SEEK_NEXT_KEY statement.

Status Codes

Values	Conditions	Notes
101	Pointer at end-of-file.	—
110	Specified key not found.	Key and record number updated.
121	Invalid file number.	Must be in range 1 through 12.
122	Improper key length.	Use key length specified when file created.
135	File is closed.	Use OPEN_KEY_FILE statement.
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).
0	Successful seek.	—

The SET_UP Statement

This statement is executed by the MIKSAM ROM when power is applied. However, it is recommended that this statement be included as part of the initialization routine in application programs. This statement should be executed only once.

Parameter:

a. ***Status code (output, numeric)*** Indicates successful buffer set up.

Sample Statements:
```
SET_UP STAT
S_U S_CODE
```

The SET_UP statement causes the ROM to perform initialization and establish buffer links. A total of ten 256-byte buffers is needed for the ROM to function properly. If sufficient memory is not available, a status code is set.

Status Codes

Values	Conditions	Notes
127	Not enough memory available.	A total of 4K bytes of user memory is required.
0	Successful set up.	—

The M_STATUS Statement

This statement provides information about the B-tree file structure used for storing and accessing key-record number pairs. This information can be helpful when considering changes in key length or file size, for example. A formula relating key file extent, key length, and number of keys is presented in appendix B.

Order of Parameters:

a. ***Status code (output, numeric)*** Indicates successful execution of the M_STATUS statement.

b. ***Key file number (input, numeric)*** Identifies the file to be accessed. Use the number assigned to the file when it is opened.

c. ***Key length (output, numeric)*** Returns the string length that must be used for the *key* parameter.

d. ***Accessible sectors (output, numeric)*** Indicates the maximum extent that can be used for storing key-record number pairs.

e. ***Free sectors (output, numeric)*** Returns the number of currently free sectors in the key file.

f. ***Tree height (output, numeric)*** Indicates the current height of the B-tree. The maximum height is seven. Refer to section 4 for specific information about key file structure.

Sample Statements:
```
M_STATUS SC,FILE_NUM,KLENGTH,TOTAL,FREE,B_TREE
M_STATUS CHECH,1,KEY_LEN,EXTENT,SECTORS,HEIGHT
```

Status Codes

Values	Conditions	Notes
121	Invalid file number.	Must be in range 1 through 12.
135	File is closed.	Use OPEN_KEY_FILE statement.
0	Successful execution.	—

Note: The *free sectors* parameter returns a value of 7 when a key file is full. If insertion is attempted, the CREATE_KEY statement returns status code 131.

Statement Summary

The parameters that must be used with each MIKSAM statement are provided in the following table.

Statements	Parameters
MAKE_KEY_FILE	***status code*** *(output, numeric)* ***key file specifier*** *(input, string)* ***key length*** *(input, numeric)* ***key file extent*** *(input, numeric)*
KILL_KEY_FILE	***status code*** *(output, numeric)* ***key file specifier*** *(input, string)*
OPEN_KEY_FILE	***status code*** *(output, numeric)* ***key file specifier*** *(input, string)* ***key file number*** *(input, numeric)*
CLOSE_KEY_FILE	***status code*** *(output, numeric)* ***key file number*** *(input, numeric)*
CREATE_KEY	***status code*** *(output, numeric)* ***key file number*** *(input, numeric)* ***record number*** *(input, numeric)* ***key*** *(input, string)*
DELETE_KEY	***status code*** *(output, numeric)* ***key file number*** *(input, numeric)* ***record number*** *(update, numeric)* ***key*** *(update, string)*
SEEK_FIRST SEEK_END	***status code*** *(output, numeric)* ***key file number*** *(input, numeric)*
SEEK_NEXT_KEY SEEK_PRIOR_KEY	***status code*** *(output, numeric)* ***key file number*** *(input, numeric)* ***record number*** *(output, numeric)* ***key*** *(output, string)*
SEEK_KEY	***status code*** *(output, numeric)* ***key file number*** *(input, numeric)* ***record number*** *(output, numeric)* ***key*** *(update, string)*
SET_UP	***status code*** *(output, numeric)*
M_STATUS	***status code*** *(output, numeric)* ***key file number*** *(input, numeric)* ***key length*** *(output, numeric)* ***accessible sectors*** *(output, numeric)* ***free sectors*** *(output, numeric)* ***tree height*** *(output, numeric)*

Parameter Listing

Depending on the MIKSAM statements you use, different parameters are necessary. The following is a comprehensive listing of the parameters used in these statements. Parameters are *numeric* unless otherwise noted.

***Accessible sectors** (output)* Returns the extent of a key file that can be accessed for storing key-record number pairs. It includes the *free sectors* parameter and any sectors currently filled with key-record number pairs. The number returned should be one less than the *key file extent* specified with the MAKE_KEY_FILE statement.

***Free sectors** (output)* Contains the number of disc sectors currently available. Refer to appendix B for information about converting a quantity of key-record number pairs to key file extent in sectors.

***Key** (input, output, update)* Passes the keys that MIKSAM files are ordered by. Any characters can be used in the key. However, the length of the string and the length defined for keys when the file is created must agree.

***Key file extent** (input)* Specifies the size of key files. The minimum extent is 13 sectors. The maximum is the number of sectors that can be stored on a volume with your system.

***Key file number** (input)* References key files once they have been opened. Only values 1 through 12 are permitted. Key file numbers are assigned with the OPEN_KEY_FILE statement.

***Key file specifier** (input)* Identifies the disc file containing the key file. The *file name* has a maximum length of 10 characters. If a non-default drive is to be accessed, then the *msus* must be included with the *file name*.

***Key length** (input, output)* Passes the length of keys to create a file with. Only values 1 through 60 can be used.

***Record number** (input, output, update)* Passes the number associated with each key value. Only values in the range 1 through 65,535 are used. If the record number is negative, then the absolute value is used. Real numbers are rounded to the nearest integer. Also, any number greater than 65,535 is set to 65,535.

***Status code** (output)* Set each time a MIKSAM statement is executed so that any execution problems can be detected. A value of zero, for example, indicates successful execution. Refer to appendix C for a complete listing.

***Tree height** (output)* Returns the height of the B-tree used for storing the key-record number pairs. A tree of height two has three levels including the root node. Tree heights can range from zero through seven. Refer to section 4 for a conceptual diagram.

> **Note:** If status code 137 is returned from a SEEK_NEXT_KEY, SEEK_PRIOR_KEY, SEEK_FIRST, SEEK_END, or SEEK_KEY statement, the pointer value can be invalid. Execution of a CREATE_KEY or DELETE_KEY statement after status code 137 is returned can corrupt a key file. Also, searches may not return the expected key-record number pair because the pointer is invalid. To correct this situation, execute a CLOSE_KEY_FILE or SET_UP statement to clear the key file buffer. Then execute an OPEN_KEY_FILE statement followed by a SEEK_KEY statement to redirect the pointer.

Section 3

Applications

Introduction

Application programs you design with MIKSAM statements can perform a variety of tasks. Since only key files are maintained with these statements, it is necessary for your application program to make the logical connection between the key files and data files used. BASIC direct access READ# and PRINT# statements are used for storing and retrieving records in data files. Sample combinations of MIKSAM statements and BASIC statements used for the following operations are provided in this section:

- Initializing files.
- Adding records.
- Deleting records.
- Updating records.
- Traversing files.
- Expanding files.
- Recovering files.

In addition to the statements, certain routines are also suggested when performing these operations. An example would be taking corrective action if a disc error occurs during key insertion or removal (page 34). These suggestions can be beneficial even if your program does not use the procedure listed. A complete application program using the suggested routines is provided in appendix D.

Monitoring free space in key files is important. When a key file is first created, a minimum extent of 13 sectors is required. The upward extent of key files is determined by your system. It is the maximum number of sectors that can be stored on the volume specified. As keys are inserted, the number of free sectors is reduced. A warning is returned when less than 12 sectors of free space remain after a key is inserted. Although keys can still be inserted, it is suggested that the key file be expanded if less than 12 sectors are available.

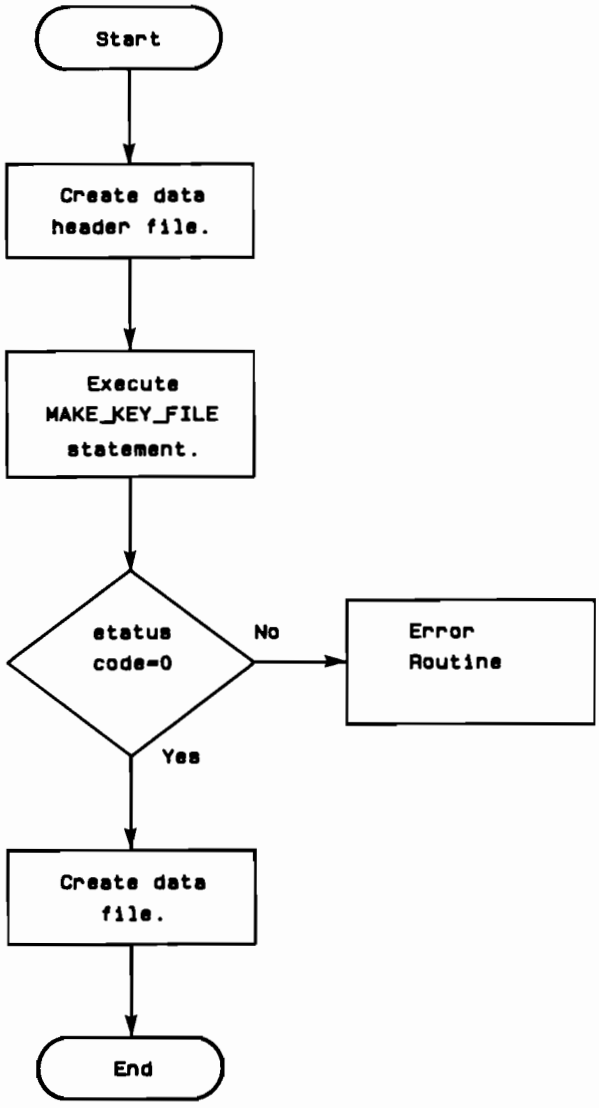
Although extremely unlikely, it is possible to reach the B-tree height limit of seven. If insertion of a key causes the tree height to exceed seven, then a warning is returned and the key is not added. It is necessary to reduce the number of keys in the file or the key length before keys can be inserted.

The MIKSAM ROM returns values to the *status code* parameter to indicate warnings. It is the programmer's responsibility to test these values and take appropriate action. Additional steps can be necessary to maintain the logical connection between key and data files.

It is recommended that all application programs be designed with a recovery routine. Procedures to recover from partial file corruption can be included in application programs. In extreme cases, a new key file must be created from the data file. The periodic backup of files is also recommended as a precautionary measure.

Initializing Files

The initialization routine can create a key file and data file. A header file containing field information about the data records can also be created. The extent specified for the key file needs to provide enough key-record number pairs for accessing the number of records created in the data file. Instructions for converting the key length and number of data file records to key file sectors are provided in appendix B.



Initialization Flowchart

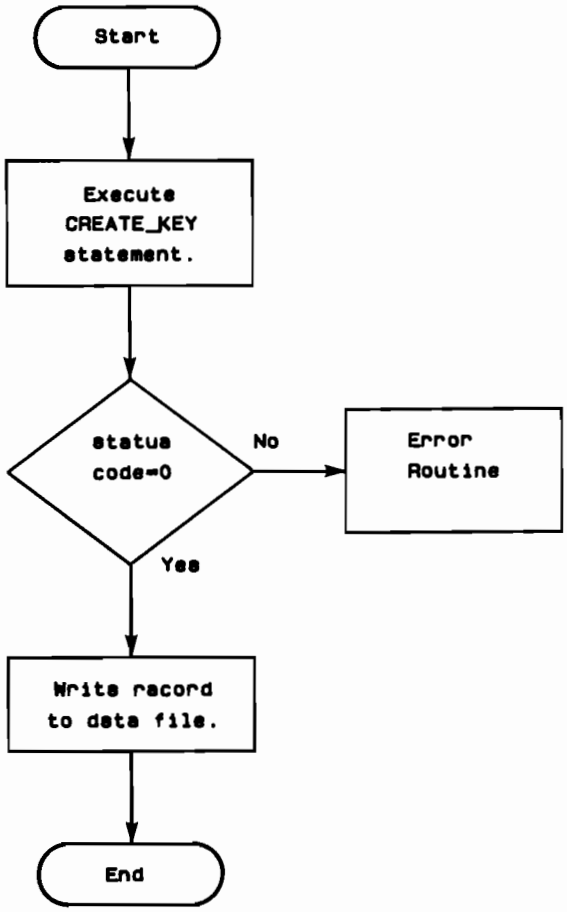
It is recommended that all fields be treated as string variables. These fields are then concatenated and written out to disc as one variable. The header file can be used for interpreting each record.

```
 :
830 DISP "--- CREATING A HEADER DATA FILE AND STORING THE FILE DEFINITIONS ---"
840 ON ERROR GOTO DISC_ERR
850    CREATE DBNAME$&"_H",4
860 OFF ERROR
870    NUM_USED=0 ! data file is initially empty
880    RECORD_MAP$[MAX_RECORDS,MAX_RECORDS]=" " ! all records are available
890 ASSIGN# 1 TO  DBNAME$&"_H"
900  PRINT# 1,1 ; NUM_USED,RECORD_MAP$
910  PRINT# 1,2 ; MAX_RECORDS,RECORD_LEN,NUM_KEYS,KEY_MAP$,KEYFIELD(),NUM_FIELDS

920  PRINT# 1,3 ! move file pointer
930  FOR I=1 TO NUM_FIELDS @ PRINT# 1 ; FIELD_NAME$(I) @ NEXT I
940  FOR I=1 TO NUM_FIELDS @ PRINT# 1 ; FIELD_LEN(I),F_BEG(I),F_END(I) @ NEXT I
950 ASSIGN# 1 TO  *
960 DISP "----- CREATING KEY FILES -----"
970 FOR J=1 TO NUM_KEYS
980 KEYFILE_SIZE=MAX_RECORDS DIV (.8*(253/(FIELD_LEN(KEYFIELD(J))+2)+1)-1)+13
990   MAKE_KEY_FILE S,DBNAME$&"_"&VAL$ (J),FIELD_LEN(KEYFIELD(J)),KEYFILE_SIZE
1000   IF S THEN DISP "MAKE_KEY_FILE ERROR. MUST EXIT." @ GOTO MKF_ERR
1010 NEXT J
1020  DISP "----- CREATING THE DATA FILE -----"
1030  ON ERROR GOTO CREATE_FILE_ERR
1040  CREATE DBNAME$,MAX_RECORDS,RECORD LEN+3
 :
```

Additions

Additions must be made to the key and data files separately. First, the record number for adding must be obtained. Then both the key and record number are inserted into the key file. Finally, information is written to the data file. After these operations are performed, the record number used in the CREATE_KEY statement must be made inaccessible to avoid overwriting the new record with future additions.



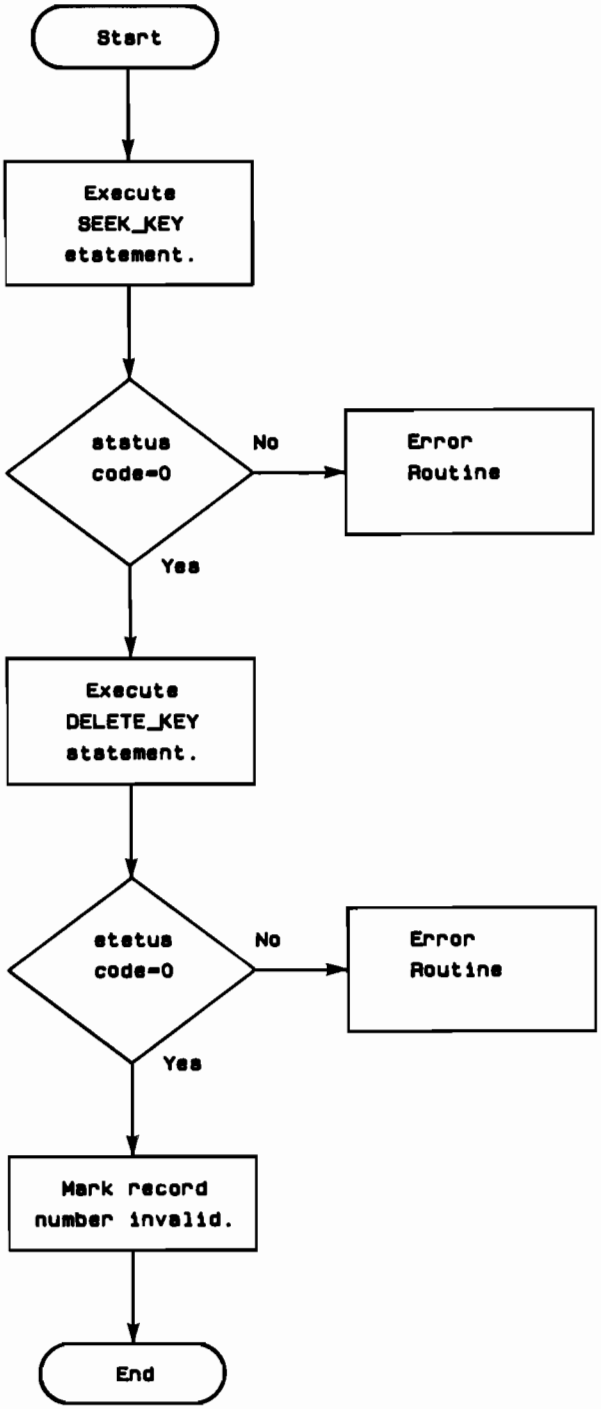
Addition Flowchart

Each addition routine should include checks for space in both data and key files. For the key file, the status code returned from the CREATE_KEY statement can be checked. When space isn't available it may be necessary to delete the key just inserted to maintain the logical connection with the data file. Messages about these actions can be displayed.

```
  :
1410 ADD_A_RECORD: IF NUM_USED=MAX_RECORDS THEN DISP "YOUR DATA BASE IS FULL.  Y
OU CANNOT ADD ANY MORE RECORDS." @ RETURN
1420   AVAIL=POS (RECORD_MAP$," ") ! find the first available record number
1430   RECORD$="" @ RECORD$[RECORD_LEN,RECORD_LEN]=" " ! set record to blank
1440   FOR F=1 TO NUM_FIELDS
1450    GOSUB GET_A_FIELD
1460   NEXT F
1470   ADDING=1
1480   GOSUB DISPLAY_RECORD
1490   GOSUB CHANGE_FIELD
1500   ADDING=0
1510 DISP "----- ADDING KEYS AND WRITING THE NEW RECORD TO THE DATA FILE -----"
1520   FOR K=1 TO NUM_KEYS
1530    NEWKEY$,KEY$(K)=RECORD$[F_BEG(KEYFIELD(K)),F_END(KEYFIELD(K))]
1540    CREATE_KEY S,K,AVAIL,KEY$(K)
1550    IF S=137 THEN GOSUB CREATE_ERROR  @ GOTO 1540
1560   NEXT K
1570   BUFNUM=2 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
1580   PRINT# BUFNUM,AVAIL ; RECORD$
1590   OFF ERROR @ IF RW_ERRFLAG THEN 1570
1600 ! mark record number AVAIL as used and increment record count
1610   NUM_USED=NUM_USED+1 @ RECORD_MAP$[AVAIL,AVAIL]="U"
1620   BUFNUM=1 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
1630   PRINT# BUFNUM,1 ; NUM_USED,RECORD_MAP$ ! update vital statistics
1640   OFF ERROR @ IF RW_ERRFLAG THEN 1620
1650   DISP "NUMBER OF ENTRIES [";NUM_USED;"] OUT OF TOTAL CAPACITY OF [";MAX_RE
CORDS;"]."
1660   IF NUM_USED=MAX_RECORDS THEN DISP "THE DATA BASE IS FULL." @ RETURN
1670   DISP "DO YOU WANT TO ADD MORE RECORDS? ENTER [Y] OR [N]."
1680   INPUT INPBUF$@ IF INPBUF$="" THEN 1670 ELSE INPBUF$=UPC$ (INPBUF$)
1690   IF POS (INPBUF$,"Y")=1 THEN 1420
1700  RETURN ! end of ADD_A_RECORD
  :
```

Deletions

Deletions require several operations since both the key and record number pairs must match for a deletion to occur. First, the record number should be retrieved with the SEEK_KEY statement. One or more SEEK_NEXT_KEY statements are needed to locate duplicate keys. Then the DELETE_KEY statement can be used to remove the key. If space in the data file is to be recycled, the record number can be added to a list of available spaces.

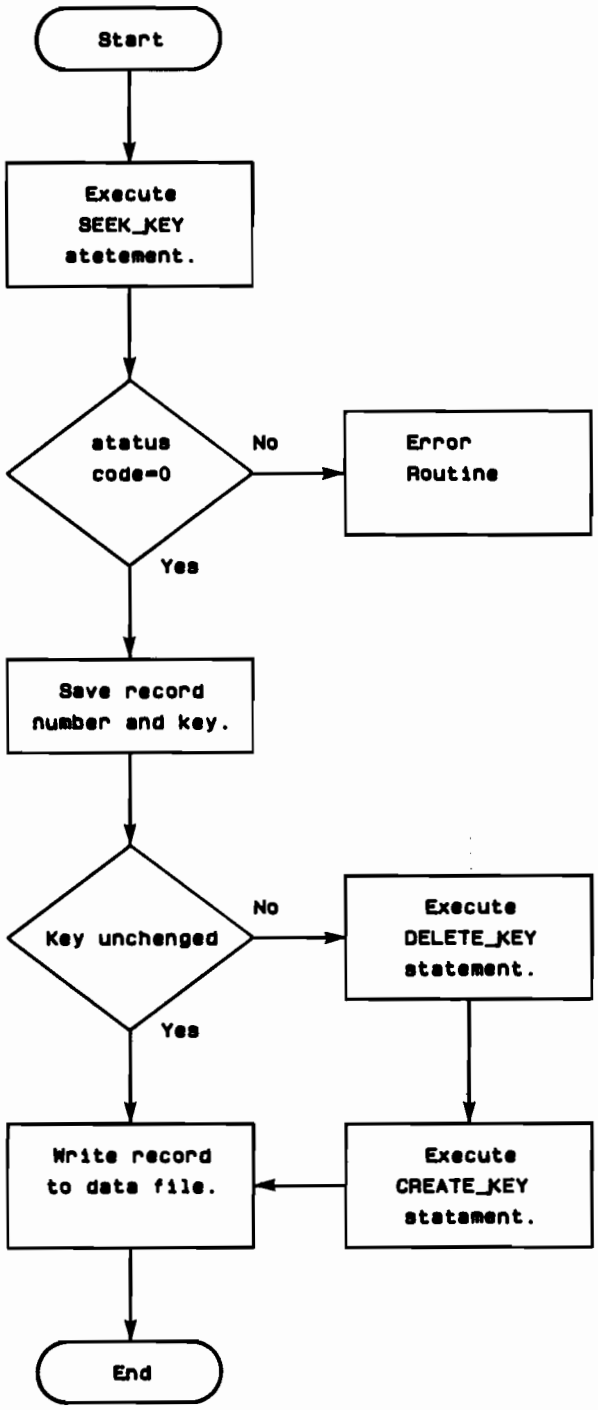


Deletion Flowchart

```
 :
2000   SEEK_KEY S,KY,RN,KEY$(KY)
2010   IF S=137 THEN GOSUB SEEK_ERROR  @ GOTO 2000
2020 IF NOT S THEN FOUND_IT
2030 !    handle cases when key not found
2040  IF S=101 THEN DISP "END OF FILE ENCOUNTERED. RE-ENTER." @ GOTO QUERY_DB
2050  IF S#110 THEN DISP "SEEK ERROR OCCURRED. MUST EXIT PROGRAM." @ END
2060  DISP "THE KEY AS SPECIFIED IS NOT IN THE KEY FILE. THE NEXT GREATER KEY RE
TRIEVED."
 :
2090 FOUND_IT: HAVE_RECORD=1 ! retrieve the desired record
2100   BUFNUM=2 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
2110   READ# BUFNUM,RN ; RECORD$
2120   OFF ERROR @ IF RW_ERRFLAG THEN 2100
2130   GOSUB DISPLAY_RECORD
2140   GOSUB SAVE_OLD_RECORD
2150 DISP "ENTER [U]PDATE, [D]ELETE, [N]EXT, [P]REVIOUS, [S]EARCH, OR [Q]UIT."
2160  INPUT A$@ IF A$="" THEN 2150 ELSE A$=UPC$ (A$) @ A$=A$[1,1]
2170  IF A$="Q" THEN RETURN
2180  IF A$="S" THEN QUERY_DB
2190 ON 1+(A$="U")+(A$="D")*2+(A$="N")*3+(A$="P")*4+(A$="S")*5 GOSUB INVAL ,UPDA
TE ,DB_DELETE ,GET_NEXT ,GET_PREV
2200 IF NOT HAVE_RECORD THEN RETURN ELSE FOUND_IT ! end of QUERY
 :
2480 DB_DELETE: ! delete the current record
2490  DISP "ARE YOU SURE YOU WANT TO DELETE THIS RECORD? ENTER [YES] OR [NO]."
2500  INPUT A$@ A$=UPC$ (A$) @ IF A$#"YES" THEN RETURN
2510  DISP "----- DELETING THE KEYS AND RECORD FROM FILES -----"
2520  FOR K=1 TO NUM_KEYS
2530   SAVEDKEY$=KEY$(K)
2540   DRN=RN @ DELETE_KEY S,K,DRN,KEY$(K)
2550   IF S=137 THEN GOSUB DELETE_ERROR  @ GOTO 2540
2560  NEXT K
2570  RECORD_MAP$[RN,RN]=" " ! RN is marked empty
2580  NUM_USED=NUM_USED-1
2590  BUFNUM=1 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
2600  PRINT# BUFNUM,1 ; NUM_USED,RECORD_MAP$ ! update vital statistics
2610  OFF ERROR @ IF RW_ERRFLAG THEN 2590
2620  SAVEDKEY$=RECORD$[F_BEG(KEYFIELD(KY)),F_END(KEYFIELD(KY))]
2630  SEEK_NEXT_KEY S,KY,RN,KEY$(KY) ! get the record after the deleted one
2640  IF S=137 THEN GOSUB NEXTSEEK_ERROR  @ GOTO 2630
2650  IF S THEN HAVE_RECORD=0 @ DISP "THERE IS NO RECORD AFTER THE DELETED RECOR
D."
2660   RETURN ! end of DB_DELETE
 :
```

Updating

Updating involves obtaining the record number of a specific record in a data file and performing a PRINT# operation to the data file. Fields to be updated must be input from the user. If the key is included in the data file, updating can involve changes to key files. To allow updates to be aborted it is necessary to preserve the original record until other records are accessed.

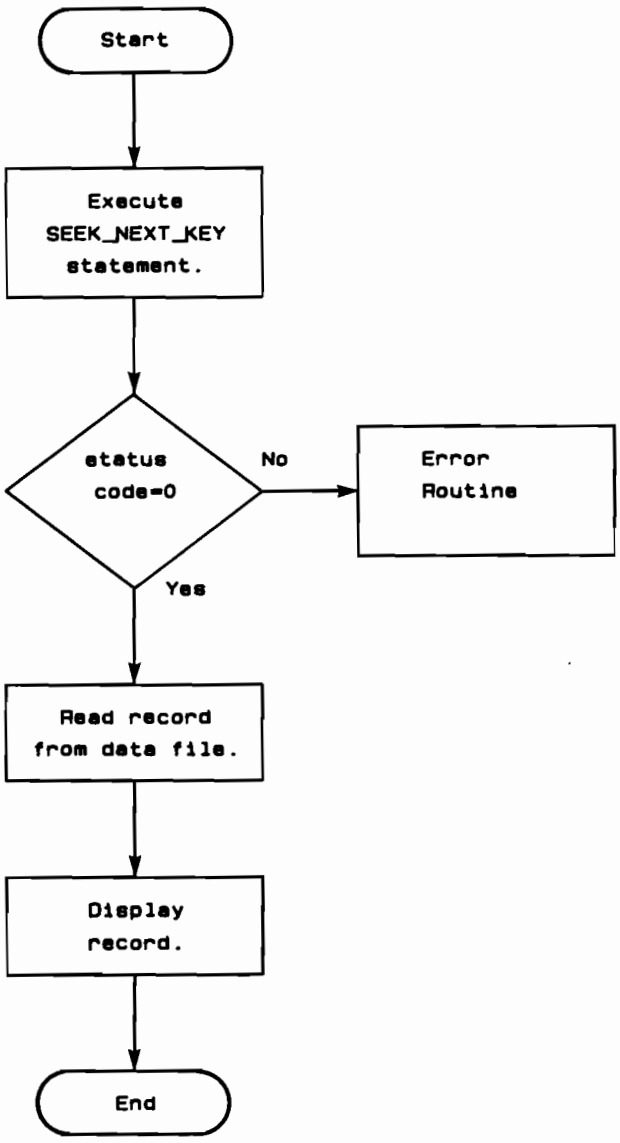


Updating Flowchart

```
  :
2240 UPDATE:
2250 GOSUB CHANGE_FIELD
2260 IF NOT CHANGED THEN RECORD$=OLD_RECORD$ @ RETURN
2270 DISP "----- UPDATING THE FILES -----"
2280 !  update key files if key values changed
2290   FOR K=1 TO NUM_KEYS
2300    NEWKEY$,KEY$(K)=RECORD$[F_BEG(KEYFIELD(K)),F_END(KEYFIELD(K))]
2310    IF KEY$(K)=OLD_KEY$(K) THEN NEXT_KEY ! key not changed
2320    SAVEDKEY$=OLD_KEY$(K)
2330    DRN=RN ! in DELETE, RN is updated so the copy should be used
2340    DELETE_KEY S,K,DRN,OLD_KEY$(K)
2350    IF S=137 THEN GOSUB UPDATE_DELETE_ERROR  @ GOTO 2330
2360    CREATE_KEY S,K,RN,KEY$(K)
2370    IF S=137 THEN GOSUB CREATE_ERROR  @ GOTO 2360
2380 NEXT_KEY: NEXT K
2390   BUFNUM=2 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
2400   PRINT# BUFNUM,RN ; RECORD$ ! update data file
2410   OFF ERROR @ IF RW_ERRFLAG THEN 2390
2420  SAVEDKEY$=RECORD$[F_BEG(KEYFIELD(KY)),F_END(KEYFIELD(KY))]
2430 SEEK_NEXT_KEY S,KY,RN,KEY$(KY) ! get next record after the updated one
2440 IF S=137 THEN GOSUB NEXTSEEK_ERROR  @ GOTO 2430
2450 IF S THEN HAVE_RECORD=0 @ DISP "THERE IS NO RECORD AFTER THE UPDATED RECORD
."
2460 RETURN ! end of UPDATE
  :
```

Traversals

Traversals are an ordered retrieval of data according to key sequence. Depending on the direction of the traversal, the SEEK_NEXT_KEY or SEEK_PRIOR_KEY statements are used. It is useful to restrict searches to certain key values when duplicate keys are present.



Traversal Flowchart

The pointer must initially be positioned at the starting key for the traversal. Use a SEEK_FIRST statement for forward traversals or a SEEK_END statement for reverse traversals. The SEEK_KEY statement can be used to position the pointer for restricted searches.

Then individual records are retrieved. For forward traversals, the SEEK_NEXT_KEY statement can be used. For reverse traversals, use the SEEK_PRIOR_KEY statement. With some applications it can be helpful to pause after each retrieval and ask the user if another record is desired.

```
  :
2070  DISP "DO YOU WANT TO GET THE RECORD FOR IT? ENTER [Y] OR [N]."
2080  INPUT A$@ A$=UPC$ (A$) @ IF POS (A$,"Y")#1 THEN GOTO 1730
2090 FOUND_IT: HAVE_RECORD=1 ! retrieve the desired record
2100   BUFNUM=2 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
2110   READ# BUFNUM,RN ; RECORD$
2120   OFF ERROR @ IF RW_ERRFLAG THEN 2100
2130   GOSUB DISPLAY_RECORD
  :
2680 GET_NEXT: ! get the record that comes after the current one in key order
2690  SAVEDKEY$=RECORD$[F_BEG(KEYFIELD(KY)),F_END(KEYFIELD(KY))]
2700  SEEK_NEXT_KEY S,KY,RN,KEY$(KY)
2710  IF S=137 THEN GOSUB NEXTSEEK_ERROR  @ GOTO 2700
2720  IF S THEN HAVE_RECORD=0 @ DISP "END OF KEY FILE REACHED."
2730 RETURN ! end of GET_NEXT
  :
```

File Expansion

The expansion of key files involves increasing the *key file extent* parameter with the MAKE_KEY_FILE statement. In most cases, both original and expanded copies of the key file must be on-line. Data file expansion is also possible. The extent of the key file and the number of records in the data file can be expanded together so that subsequent key insertions won't exhaust space in the data file.

Care should be taken to preserve the logical relationship between key and data files. Depending on how available space in the data file is managed, different approaches can be needed for expanding it. If a linked list is used, it must be modified with the increased number of records and any null spaces. Refer to **Key File Recovery** for additional information.

Key File Recovery

Certain situations can cause key files to become corrupt. For example, interruptions made while key file sectors are written out to disc can make the pointer invalid (page 21). To determine the specific cause, the BASIC ERRN statement can be used. Here are some typical situations:

- Disc door open, ERRN 130.
- Write-protect tab on disc, ERRN 60.

Refer to the *HP-86/87 Operating and BASIC Programming Manual* for a detailed listing of ERRN error codes.

If the data file is not corrupt, it can be used to recover the key file. The key value must be contained in the data file. The EXTEND/RECOVER subroutine listed on pages 59 through 61 reads a data file, extracts key fields, and creates key files from the key field data.

MIKSAM for Advanced Programmers

After you implement a data base management application with statements in a BASIC program, you can add records to a data file. In each READ# or PRINT# operation, care must be taken to maintain the validity of the key files. The connection between data files and key files is purely logical.

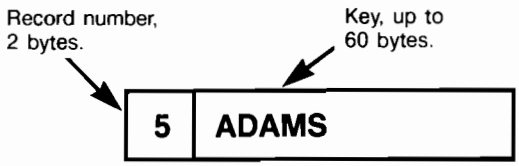
The ROM does not link keys with data files. It is the responsibility of the programmer to verify that the record number associated with a key points to the correct record in a data file. The ROM does not use BASIC statements.

Although duplicate keys are allowed, it is up to the programmer to locate them in the key files with the SEEK_NEXT_KEY statement. Only the first key with the specified value is returned in a search using the SEEK_KEY statement. No constraints are placed on the record numbers associated with these keys. The programmer must maintain the correspondence between keys and record numbers in the key file and record numbers of specific records in the data file.

File Structure

The ROM creates a B-tree for storing keys and record numbers. Each key file contains information for only one key. Multiple keys require multiple key files.

The length of keys is defined when each key file is created. Lengths of 1 through 60 bytes can be used. Keys are interpreted as string values. The ordering of keys is determined by the ASCII collating sequence. The following diagram shows the structure of one key-record number pair in a leaf node.

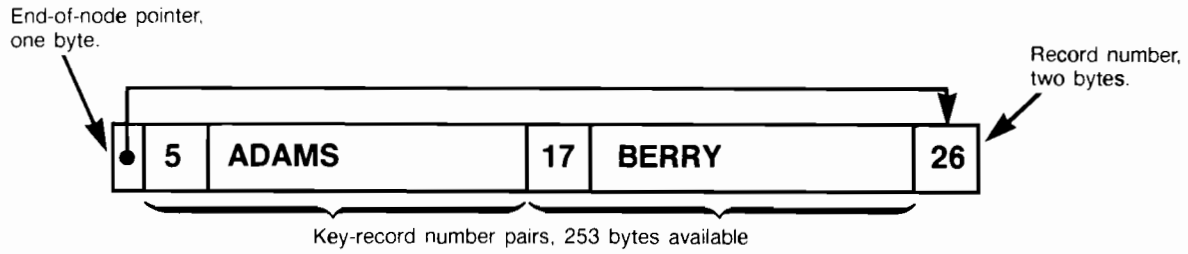


Key-Record Number Pair

Corresponding pairs in non-leaf nodes contain a pointer value instead of a record number. The pointer is used for locating nodes that contain key-pointer or key-record number pairs with lower key values.

The key length specified when a file is created and the bytes needed for storing the key are the same. The record number or pointer value associated with the key is always two bytes long. When a key is accessed, the sector containing the key is read into computer memory. This sector is equivalent to one node of the B-tree. A formula relating key length and key file sectors is provided in appendix B.

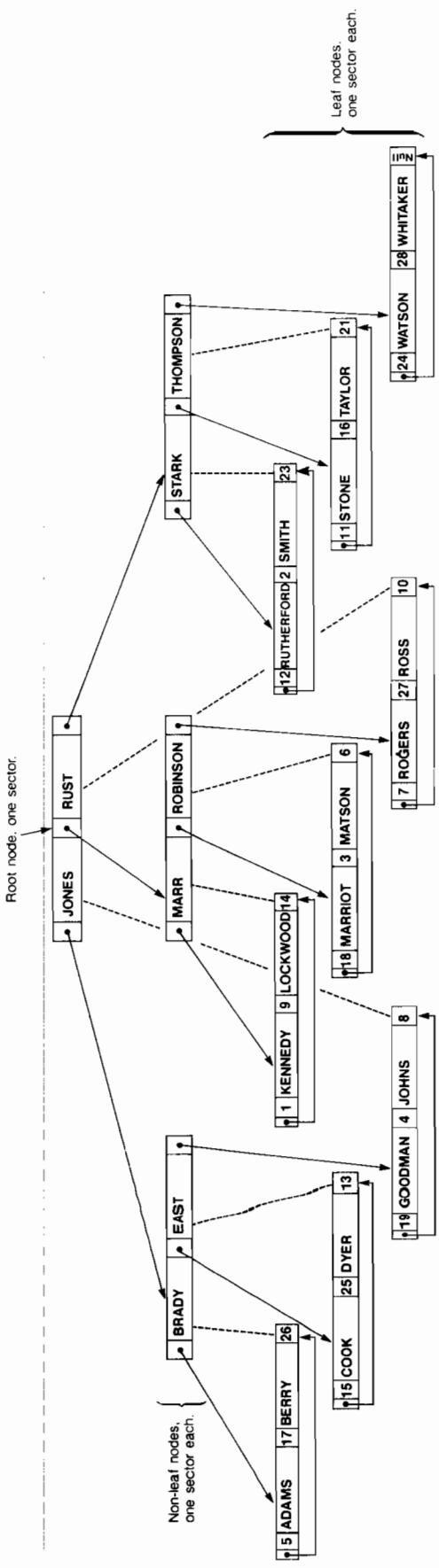
The maximum number of keys that can be stored in a node is determined by the key length. In addition to the bytes needed for each key-record number or key-pointer value, all nodes also use the leftmost byte as an end-of-node pointer. The record number of a key in a non-leaf node is stored in the rightmost two bytes of the rightmost leaf node of the left subtree of the key. The extent of each node is one sector and is therefore 256 bytes:



Leaf Node

The structure used for key files is a B-tree of degree *n*, where *n* is the number of key-record number pairs that can be stored on a leaf node. The B-tree consists of a root node, non-leaf nodes, and the leaf nodes. The height of the B-tree is the number of non-root levels in the tree. The following B-tree has a height of two. The addition of leaf nodes can increase the number of non-leaf nodes which can increase the height of the tree. The maximum height of the B-tree is seven.

The ROM also creates a header record for each key file that contains system information such as the current height of the B-tree, the next available node, the key length, the maximum and current extents of the file, and the open status of the file. Each header record requires one sector on disc. Information in the header record is accessed with the M_STATUS statement.



B-tree Structure

Performance

The frequency of disc input/output operations determine how quickly you can locate different keys. In general, the shorter the key is, the faster keys can be accessed. This is because more key-record number pairs can be stored in each node. After a specific leaf node is read into computer memory, any keys stored on it can be accessed without disc operations.

As a safeguard, output to the key file is handled directly. This means that all insertions of keys are written out to disc immediately. Direct output minimizes the chance of key file corruption if a power interruption occurs. This feature also means that it is not necessary to write out to disc when the key file is closed.

To insure uniformity in key access times, it is necessary to keep the B-tree balanced. This is done automatically. Whenever a key is inserted or deleted, a check is made to see if rebalancing is necessary. These two situations trigger the rebalancing routine:

- A key is inserted onto a leaf node that is already full.
- A less than half-full leaf node results from deleting a key.

The B-tree balancing routine can require several disc input/output operations. The shortest routine involves placing a key-record number pair on an adjacent leaf node. Adding another level to the B-tree can be necessary when keys are inserted in a certain order.

Appendix A

Maintenance, Service, and Warranty

Maintenance

The MIKSAM ROM doesn't require maintenance. However, there are several areas of caution that you should be aware of. They are:

WARNING: Do not place fingers, tools, or other foreign objects into the plug-in ports. Such actions can result in minor electrical shock hazard and interference with some pacemaker devices. Damage to plug-in port contacts and the computer's internal circuitry can also result.

CAUTION: Always switch off the HP-86/87 and any peripherals involved when inserting or removing modules. Use only plug-in modules designed by Hewlett-Packard specifically for the HP-86/87. Failure to do so could damage the module, the computer, or the peripherals.

CAUTION: If a module or ROM drawer jams when inserted into a port it may be upside down or designed for another port. Attempting to force it can damage the computer or the module. Remove the module carefully and reinsert it.

CAUTION: Do not touch the spring-finger connectors on the ROM with your fingers or other objects. Static discharge could damage the electrical components.

CAUTION: Handle all ROMs very carefully while they are out of the ROM drawer. Do not insert any objects in the contact holes on the ROM drawer. Always keep the protective cap in place over the ROM drawer contacts while the ROM is not plugged into the ROM drawer. Failure to observe these cautions can result in damage to the ROM or ROM drawer.

For instructions on how to insert and remove the ROM and ROM drawer, please refer to the instruction sheet supplied with the ROM drawer or section 2 of your computer's introductory manual.

Service

If at any time you suspect the MIKSAM ROM or the ROM drawer to be malfunctioning, do the following:

1. Turn the computer and all peripherals off. Disconnect all peripherals and remove the ROM drawer from the HP-86/87 port. Turn the computer back on. If it doesn't respond or displays `ERROR 23 : SELF-TEST`, the computer requires service.
2. Turn the computer off. Insert the ROM drawer, with the MIKSAM ROM installed, into any port. Turn the computer on again.
   - If the cursor does not appear, the system is not operating properly. To help determine what is causing the improper operation, repeat step 2 with the ROM drawer inserted in a different port, both with the MIKSAM ROM installed in the ROM drawer and with the ROM removed from the ROM drawer.

- If `Error 109 : MIKSAM ROM` is displayed, indicating that the ROM is not operating properly, turn the computer off and try the ROM in another ROM drawer slot. This will help you determine if particular slots in the ROM drawer are malfunctioning, or if the ROM itself is malfunctioning.

3. Refer to **Obtaining Repair Service** for information on how to obtain repair service for the malfunctioning device.

Radio/Television Interference Statement

The MIKSAM ROM uses radio frequency energy and may cause interference to radio and television reception. The ROM has been type-tested and found to comply with limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of the Federal Communications Commission Rules. These specifications provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If the ROM does cause interference to radio or television, which can be determined by turning the computer on and off with the ROM installed and with the ROM removed, you can try to eliminate the interference problem by doing one or more of the following:

- Reorient the receiving antenna.
- Change the position of the computer with respect to the receiver.
- Move the computer away from the receiver.
- Plug the computer into a different outlet so that the computer and the receiver are on different branch circuits.

If necessary, consult an authorized HP dealer or an experienced radio/television technician for additional suggestions. You may find the following booklet, prepared by the Federal Communications Commission, helpful: *How to Identify and Resolve Radio/TV Interference Problems.* This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock No. 004-000-00345-4.

Warranty Information

The complete warranty statement is included in the information packet shipped with your ROM. Additional copies can be obtained from any authorized Hewlett-Packard dealer.

If you have any questions concerning this warranty, please contact:

In the U.S.: One of the six Field Repair Centers listed on the service information sheet packaged with your owner's documentation.

In other countries: Contact your nearest sales and service facility. If you are unable to contact that facility, please contact:

In Europe:

Hewlett-Packard
7, rue du Bois-du-Lan
P. O. Box
CH-1217 Meyrin 2
Geneva
Switzerland
Tel. (022) 82 70 00

Other countries:

Hewlett-Packard Intercontinental
3495 Deer Creek Rd.
Palo Alto, California 94304
U.S.A.
Tel. (415) 857-1501

Obtaining Repair Service

Not all Hewlett-Packard facilities offer service for the HP-86/87 and its peripherals. For information on service in your area, contact your nearest authorized HP dealer or the nearest Hewlett-Packard sales and service center.

If your computer system malfunctions and repair is required, you can help assure efficient service by providing this information:

a. A description of the configuration of the HP-86/87, exactly as it was at the time of malfunction.

b. A brief yet specific description of the malfunction symptoms for service personnel.

c. Printouts or any other materials that illustrate the problem area.

d. A copy of the sales slip or other proof of purchase to establish the warranty coverage period.

Serial Numbers

Each Series 80 computer carries an individual serial number plate on the rear panel. We recommend that owners keep a separate record of this number. Should your unit be lost or stolen, the serial number is often necessary for tracing and recovery, as well as for insurance claims. Hewlett-Packard doesn't maintain records of individual owner's names and unit serial numbers.

General Shipping Instructions

Should you ever need to ship any portion of your HP-86/87 system, be sure that it is packed in a protective package to avoid in-transit damage. Use the original shipping case if possible. Hewlett-Packard suggests that the customer always insure shipments. Any customs or duty charges are also the customer's responsibility.

Appendix B

Extent Conversions

When a key file is created, the extent of the file is specified in sectors. The formula below can be used to convert the number of keys needed to *key file extent* in sectors—used in the MAKE_KEY_FILE statement. In most cases, the number of keys needed and the number of records in the associated data file are identical. This formula is only an approximation. You may be able to store more key-record number pairs than what the formula indicates. The actual number depends on the key values and the order in which keys are inserted.

key file extent = ***number of keys*** DIV (0.8 * (253 / (***key length*** + 2) + 1) - 1) + 13

number of keys: The number of key-record number pairs that can be inserted into the key file. The maximum is 65,535 pairs.

key length: The number of bytes in the key. The acceptable range is 1 through 60 bytes.

Appendix C

Status Codes

Status codes are returned by each MIKSAM statement. These codes provide information about the execution of statements. The status codes that can be returned by each statement are shown in the **Statement/Status Code Summary.** The **Status Code Listing** (next page) describes the meaning of each status code. For specific information about the statements, refer to section 2.

Statement/Status Code Summary

Statements	Codes	Statements	Codes	Statements	Codes
MAKE_KEY_FILE	0 120 122 125 130 132 137	CREATE_KEY	0 102 121 122 128 131 135 136 137	SEEK_NEXT_KEY	0 101 121 122 135 137
KILL_KEY_FILE	0 120 126 133 134 137	DELETE_KEY	0 111 121 122 128 135 137	SEEK_PRIOR_KEY	0 101 121 122 135 137
OPEN_KEY_FILE	0 103 120 121 126 133 134 137	M_STATUS	0 121 135	SEEK_KEY	0 101 110 121 122 135 137
		SEEK_FIRST	0 121 135 137		
CLOSE_KEY_FILE	0 121 135 137	SEEK_END	0 121 135 137	SET_UP	0 127

Status Code Listing

Values	Conditions	Notes
0	Successful execution.	—
101	End-of-file encountered.	Record number and key unchanged.
102	Space low—less than 12 sectors remain.	Increase extent of key file.
103	File now open but not closed properly.	Check for corrupt data.
110	Specified key not found.	Key and record number updated.
111	Key-record number pair not found.	If key is in file, check associated record number.
120	Incorrect device address.	Check *msus* and address of drive.
121	Invalid file number.	File is open or file number is not in range 1 through 12.
122	Invalid key length.	Use length specified during creation (1 through 60 bytes).
125	Invalid extent.	Must be at least 13 sectors.
126	File not PKEY type.	Obtain directory of disc with ROM installed.
127	Not enough memory available.	A total of 4K bytes of memory is required.
128	Invalid record number.	Evaluates to zero.
130	Not enough room for file.	Purge unneeded files.
131	No space in key file.	Increase key file extent.
132	File already exists.	Must change the *file specifier* or delete the existing file.
133	File in parameter is not on-line.	Check *msus*.
134	File is open.	—
135	File is closed.	Use OPEN_KEY_FILE statement.
136	B-tree height is at maximum of seven.	Reduce key length or number of keys in file.
137	Disc access error.	Disc not initialized, disc door open, or select code incorrect (page 21).

Appendix D

Sample Application Program

The following program can be executed on an HP-86/87 computer with the MIKSAM ROM installed. The first part of the program allows you to create a data base consisting of:

a. A random access data file where records are stored.

b. A header file for file definitions and current dynamic statistics about the data file.

c. Up to three MIKSAM key files for each data file.

After files are stored on disc, the second part of the program can be used. These operations are possible:

a. Adding records.

b. Retrieving records with specific key values.

c. Traversing data file records in forward or reverse key order.

d. Updating the current or most recently retrieved record.

e. Deleting the current record.

Expansion or recovery of key files is also possible. The limits of the data and key files created can be changed by modifying program constants (shown below).

```
10 ! constants for MIKSAM application program
20   MAXFIELDS=10 !      maximum number of fields
30   MAXFLEN=60 !        maximum field length
40   MAXRLEN=600 !        maximum length of record
50   MAXKEYS=3 !         maximum number of key fields
60   MAXFNAME=12 !       maximum length of field names
70   MAXDBSIZE=100 !     maximum number of records in the data base
80 ! if constants are changed, string and numeric arrays must be
90 ! dimensioned to accommodate the new data base specifications
140 PAGESIZE 24
150 OPTION BASE 1
160 DIM FIELD_NAME$(10)[12],FIELD_LEN(10),KEYFIELD(3),DBNAME$[10],ERRFILE$[10]
170 DIM KEY_MAP$[10],RECORD_MAP$[100],INPBUF$[80],KEY$(3)[60],OLD_KEY$(3)[60]
180 DIM NEWKEY$[60]
190 DIM RECORD$[600],OLD_RECORD$[600],SAVEDKEY$[60]
200 INTEGER NUM_FIELDS,FIELD_COUNT,NUM_KEYS,MAX_RECORDS,NUM_USED
210 INTEGER RECORD_LEN,F_BEG(10),F_END(10),END_FIELD,FILE_OPENED
220 FILE_OPENED,ADDING=0 ! set flags
230 CLEAR @ FOR I=1 TO 5 @ DISP @ NEXT I
240 DISP TAB (10);"**** WELCOME TO THE MIKSAM APPLICATION PROGRAM ****"
250 DISP @ DISP TAB (10);"MIKSAM by "&MIKSAM
260 DISP @ DISP "PRESS [CONT] TO START." @ PAUSE
270 SET_UP S ! initialize MIKSAM buffer
280 CLEAR @ DISP "** PLEASE DO NOT SWAP DISCS UNLESS YOU ARE SO PROMPTED **"
290 DISP "ENTER THE DATA BASE NAME (MAX. 8 CHARS.) OR ENTER [END] TO EXIT."
300 INPUT INPBUF$@ IF INPBUF$="" THEN 290 ! null string illegal
310   IF LEN (INPBUF$)<9 THEN DBNAME$=INPBUF$ ELSE DBNAME$=INPBUF$[1,8]
320   INPBUF$=UPC$ (INPBUF$)
330   IF INPBUF$="END" OR INPBUF$="E" THEN DB_END
340   SET_UP S ! clear MIKSAM buffer for a new file
350 CLEAR
360 DISP "ENTER [C]REATE, [A]CCESS, [E]XTEND, [R]ECOVER A DATA BASE OR [Q]UIT."
370   INPUT INPBUF$@ IF INPBUF$="" THEN 360 ! null string illegal
```

```
380   INPBUF$=UPC$ (INPBUF$) @ A$=INPBUF$[1,1]
390 ON 1+(A$="C")+(A$="A")*2+(A$="E")*3+(A$="R")*4+(A$="Q") GOTO 440,400,410,420
,430
400   GOSUB CREATE_DB  @ GOTO 280
410   GOSUB ASK_DB  @ GOTO 280
420   EXTENDING=1 @ GOSUB EXTEND_RECOVER  @ GOTO 280
430   EXTENDING=0 @ GOSUB EXTEND_RECOVER  @ GOTO 280
440 DISP "PLEASE ENTER A VALID RESPONSE." @ GOTO 360
450 DB_END: GOSUB CLOSE_FILES
460 DISP " ******** END OF THE MIKSAM APPLICATION PROGRAM ********** "
470 END
490 CREATE_DB: CLEAR
500 DISP "ENTER THE MAXIMUM NUMBER OF RECORDS IN THIS DATA BASE. MUST BE <=";MAX
DBSIZE;"."
510  INPUT MAX_RECORDS@ IF MAX_RECORDS>MAXDBSIZE THEN 500
520 DISP "ENTER THE NUMBER OF FIELDS IN A RECORD. MUST BE <=";MAXFIELDS;"."
530  INPUT NUM_FIELDS@ IF NUM_FIELDS>MAXFIELDS THEN 520
540 !    define each field of the record in the data base
550 NUM_KEYS,RECORD_LEN,END_FIELD,NUM_USED=0 ! initialize file variables
560 FOR K=1 TO MAXKEYS @ KEYFIELD(K)=0 @ NEXT K ! initialize key fields as null
570 FOR I=1 TO NUM_FIELDS
580   DISP "ENTER THE NAME OF FIELD #";I;" OF UP TO ";MAXFNAME;" CHARACTERS."
590   INPUT INPBUF$
600   IF LEN (INPBUF$)<= MAXFNAME THEN FIELD_NAME$(I)=INPBUF$ ELSE FIELD_NAME$(I
)=INPBUF$[1,MAXFNAME]
610   DISP "ENTER THE LENGTH OF THIS FIELD. MUST BE <=";MAXFLEN;"."
620   INPUT FIELD_LEN(I)
630   IF FIELD_LEN(I)>MAXFLEN THEN 610
640   F_BEG(I)=END_FIELD+1 @ F_END(I)=F_BEG(I)+FIELD_LEN(I)-1
650   END_FIELD=F_END(I)
660   RECORD_LEN=RECORD_LEN+FIELD_LEN(I)
670   IF RECORD_LEN>MAXRLEN THEN DISP "TOTAL LENGTH OF A RECORD MUST BE <=";MAXR
LEN;"." @ GOTO 550 ! restart definition
680   KEY_MAP$[I,I]="N" ! set this field to non-key
690   IF NUM_KEYS=MAXKEYS THEN 750 ! cannot be more than maximum key fields
700   DISP "IS THIS FIELD A KEY FIELD? ENTER [Y] OR [N]."
710   INPUT INPBUF$@ INPBUF$=UPC$ (INPBUF$)
720   KEY_MAP$[I,I]="N"
730   IF POS (INPBUF$,"Y")#1 THEN 750
740   NUM_KEYS=NUM_KEYS+1 @ KEYFIELD(NUM_KEYS)=I @ KEY_MAP$[I,I]="Y" ! field I
is a key
750 NEXT I
760 IF NOT NUM_KEYS THEN DISP "THERE MUST BE AT LEAST ONE KEY FIELD. YOU MUST RE
DEFINE THE FILE." @ GOTO 500
770 GOSUB SHOW_DEFINITION  ! all fields have been defined
780 DISP "DO YOU WANT TO MAKE CHANGES IN THIS DEFINITION? ENTER [Y] OR [N]."
790 INPUT INPBUF$@ IF INPBUF$="" THEN 780 ELSE INPBUF$=UPC$ (INPBUF$)
800 IF POS (INPBUF$,"Y")=1 THEN 500 ! redefine the data base
810 DISP "PLEASE INSERT A DISC IN THE DEFAULT DRIVE FOR STORING THE DATA BASE."
820 DISP "PRESS [CONT] WHEN DONE." @ PAUSE
830 DISP "--- CREATING A HEADER DATA FILE AND STORING THE FILE DEFINITIONS ---"
840 ON ERROR GOTO DISC_ERR
850    CREATE DBNAME$&"_H",4
860 OFF ERROR
870    NUM_USED=0 ! data file is initially empty
880    RECORD_MAP$[MAX_RECORDS,MAX_RECORDS]=" " ! all records are available
890 ASSIGN# 1 TO  DBNAME$&"_H"
900  PRINT# 1,1 ; NUM_USED,RECORD_MAP$
910  PRINT# 1,2 ; MAX_RECORDS,RECORD_LEN,NUM_KEYS,KEY_MAP$,KEYFIELD(),NUM_FIELDS

920  PRINT# 1,3 ! move file pointer
930  FOR I=1 TO NUM_FIELDS @ PRINT# 1 ; FIELD_NAME$(I) @ NEXT I
940  FOR I=1 TO NUM_FIELDS @ PRINT# 1 ; FIELD_LEN(I),F_BEG(I),F_END(I) @ NEXT I
950 ASSIGN# 1 TO  *
960 DISP "----- CREATING KEY FILES -----"
970 FOR J=1 TO NUM_KEYS
980 KEYFILE_SIZE=MAX_RECORDS DIV (.8*(253/(FIELD_LEN(KEYFIELD(J))+2)+1)-1)+13
990   MAKE_KEY_FILE S,DBNAME$&"_"&VAL$ (J),FIELD_LEN(KEYFIELD(J)),KEYFILE_SIZE
1000   IF S THEN DISP "MAKE_KEY_FILE ERROR. MUST EXIT." @ GOTO MKF_ERR
1010 NEXT J
```

```
1020  DISP "----- CREATING THE DATA FILE -----"
1030  ON ERROR GOTO CREATE_FILE_ERR
1040  CREATE DBNAME$,MAX_RECORDS,RECORD_LEN+3
1050  OFF ERROR
1060 DISP "----- THE DATA BASE IS SUCCESSFULLY CREATED AND INITIALIZED -----"
1070 RETURN
1080 DISC_ERR: OFF ERROR @ DISP "A DISC ERROR HAS OCCURRED." @ GOTO 810
1090 CREATE_FILE_ERR: OFF ERROR
1100   DISP "ERROR IN CREATING THE DATA FILE. MUST EXIT THE PROGRAM."
1110 MKF_ERR: FOR K=1 TO J
1120            KILL_KEY_FILE S,DBNAME$&"_"&VAL$ (K)
1130          NEXT K
1140  DISP "PREMATURE TERMINATION OF THE PROGRAM DUE TO FILE ERROR."
1150 END ! CREATE_DB
1170 ASK_DB: GOSUB OPEN_HEADER  ! read the header record
1180    IF NOT FILE_OPENED THEN RETURN ! if header read unsuccessful
1190 DISP "----- OPENING KEY FILES -----"
1200   FOR J=1 TO NUM_KEYS
1210    OPEN_KEY_FILE S,DBNAME$&"_"&VAL$ (J),J
1220    IF NOT (S=0 OR S=103) THEN DISP "OPEN_KEY_FILE ERROR-MUST EXIT." @ END
1230    KEY$(J)[FIELD_LEN(KEYFIELD(J)),FIELD_LEN(KEYFIELD(J))]=" " ! set length
of key
1240   NEXT J
1250 DISP "----- OPENING THE DATA FILE -----"
1260   ASSIGN# 2 TO  DBNAME$
1270 GOSUB SHOW_DEFINITION
1280 DISP "ENTER [A]DD, [S]EARCH RECORDS, OR [Q]UIT."
1290 INPUT INPBUF$@ IF INPBUF$="" THEN 1280 ELSE INPBUF$=UPC$ (INPBUF$)
1300 IF POS (INPBUF$,"Q")=1 THEN GOSUB CLOSE_FILES  @ RETURN
1310 IF POS (INPBUF$,"A")=1 THEN GOSUB ADD_A_RECORD  @ GOTO 1280
1320 IF NOT NUM_USED THEN DISP "THERE ARE NO RECORDS IN THIS DATA BASE. YOU CAN
ONLY ADD RECORDS." @ GOTO 1280
1330 IF POS (INPBUF$,"S")=1 THEN GOSUB QUERY_DB
1340 GOTO 1280 ! in case of invalid response
1350 CHECK_FILE: OFF ERROR @ FILE_OPENED=0
1360  DISP "THERE IS A FILE ERROR.  CAN'T OPEN THE FILE."
1370  DISP "IS THE FILE NAME [";DBNAME$;"] CORRECT? ENTER [Y] OR [N]."
1380  INPUT A$@ A$=UPC$ (A$)
1390  IF POS (A$,"Y")=1 THEN ASK_DB ELSE RETURN ! end of ASK_DB
1410 ADD_A_RECORD: IF NUM_USED=MAX_RECORDS THEN DISP "YOUR DATA BASE IS FULL.  Y
OU CANNOT ADD ANY MORE RECORDS." @ RETURN
1420   AVAIL=POS (RECORD_MAP$," ") ! find the first available record number
1430   RECORD$="" @ RECORD$[RECORD_LEN,RECORD_LEN]=" " ! set record to blank
1440   FOR F=1 TO NUM_FIELDS
1450    GOSUB GET_A_FIELD
1460   NEXT F
1470   ADDING=1
1480   GOSUB DISPLAY_RECORD
1490   GOSUB CHANGE_FIELD
1500   ADDING=0
1510 DISP "----- ADDING KEYS AND WRITING THE NEW RECORD TO THE DATA FILE -----"
1520   FOR K=1 TO NUM_KEYS
1530    NEWKEY$,KEY$(K)=RECORD$[F_BEG(KEYFIELD(K)),F_END(KEYFIELD(K))]
1540    CREATE_KEY S,K,AVAIL,KEY$(K)
1550    IF S=137 THEN GOSUB CREATE_ERROR  @ GOTO 1540
1560   NEXT K
1570   BUFNUM=2 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
1580   PRINT# BUFNUM,AVAIL ; RECORD$
1590   OFF ERROR @ IF RW_ERRFLAG THEN 1570
1600 ! mark record number AVAIL as used and increment record count
1610   NUM_USED=NUM_USED+1 @ RECORD_MAP$[AVAIL,AVAIL]="U"
1620   BUFNUM=1 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
1630   PRINT# BUFNUM,1 ; NUM_USED,RECORD_MAP$ ! update vital statistics
1640   OFF ERROR @ IF RW_ERRFLAG THEN 1620
1650   DISP "NUMBER OF ENTRIES [";NUM_USED;"] OUT OF TOTAL CAPACITY OF [";MAX_RE
CORDS;"]."
1660   IF NUM_USED=MAX_RECORDS THEN DISP "THE DATA BASE IS FULL." @ RETURN
1670   DISP "DO YOU WANT TO ADD MORE RECORDS? ENTER [Y] OR [N]."
1680   INPUT INPBUF$@ IF INPBUF$="" THEN 1670 ELSE INPBUF$=UPC$ (INPBUF$)
1690   IF POS (INPBUF$,"Y")=1 THEN 1420
```

```
1700  RETURN ! end of ADD_A_RECORD
1720 QUERY_DB: ! traversal, search, update, deletion
1730   DISP "CHOOSE A KEY FIELD WHICH YOU WANT TO USE TO SEARCH RECORDS."
1740   DISP "THE FOLLOWING FIELDS ARE KEYS."
1750   DISP TAB (2);"KEY #";TAB (15);"FIELD NAME";TAB (30);"FIELD LENGTH"
1760   FOR K=1 TO NUM_KEYS
1770    DISP TAB (4);K;TAB (12);FIELD_NAME$(KEYFIELD(K));TAB (35);FIELD_LEN(KEYF
IELD(K))
1780   NEXT K
1790   OFF ERROR
1800   DISP "ENTER A KEY NUMBER."
1810   INPUT A$
1820   ON ERROR GOTO 1790
1830   KY=VAL (A$)
1840   OFF ERROR
1850   IF KY>NUM_KEYS THEN DISP "MUST BE ONE OF THE KEY FIELDS.";@ GOTO 1800
1860   IF KEY_MAP$[KEYFIELD(KY),KEYFIELD(KY)]#"Y" THEN DISP "IT IS NOT A KEY FIE
LD. ";@ GOTO 1800
1870 DISP "ENTER THE KEY VALUE OR TYPE [F]IRST OR [L]AST TO GET THE FIRST OR LAS
T RECORD."
1880   INPUT INPBUF$@ IF INPBUF$="" THEN 1870 ELSE INPBUF$=UPC$ (INPBUF$)
1890   IF NOT (INPBUF$="F" OR INPBUF$="FIRST") THEN 1930
1900    SEEK_FIRST S,KY @ IF S=137 THEN 1920
1910    SEEK_NEXT_KEY S,KY,RN,KEY$(KY)
1920    IF S=137 THEN GOSUB FIRSTSEEK_ERROR  @ GOTO 1910 ELSE 2020
1930   IF NOT (INPBUF$="L" OR INPBUF$="LAST") THEN 1970
1940    SEEK_END S,KY @ IF S=137 THEN 1960
1950    SEEK_PRIOR_KEY S,KY,RN,KEY$(KY)
1960    IF S=137 THEN GOSUB ENDSEEK_ERROR  @ GOTO 1940 ELSE 2020
1970   IF LEN (INPBUF$)>FIELD_LEN(KY) THEN INPBUF$=INPBUF$[1,FIELD_LEN(KY)]
1980   KEY$(KY)[1,FIELD_LEN(KEYFIELD(KY))]=INPBUF$
1990   SAVEDKEY$=KEY$(KY)
2000   SEEK_KEY S,KY,RN,KEY$(KY)
2010   IF S=137 THEN GOSUB SEEK_ERROR  @ GOTO 2000
2020 IF NOT S THEN FOUND_IT
2030 !    handle cases when key not found
2040  IF S=101 THEN DISP "END OF FILE ENCOUNTERED. RE-ENTER." @ GOTO QUERY_DB
2050  IF S#110 THEN DISP "SEEK ERROR OCCURRED. MUST EXIT PROGRAM." @ END
2060  DISP "THE KEY AS SPECIFIED IS NOT IN THE KEY FILE. THE NEXT GREATER KEY RE
TRIEVED."
2070  DISP "DO YOU WANT TO GET THE RECORD FOR IT? ENTER [Y] OR [N]."
2080  INPUT A$@ A$=UPC$ (A$) @ IF POS (A$,"Y")#1 THEN GOTO 1730
2090 FOUND_IT: HAVE_RECORD=1 ! retrieve the desired record
2100   BUFNUM=2 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
2110   READ# BUFNUM,RN ; RECORD$
2120   OFF ERROR @ IF RW_ERRFLAG THEN 2100
2130   GOSUB DISPLAY_RECORD
2140   GOSUB SAVE_OLD_RECORD
2150 DISP "ENTER [U]PDATE, [D]ELETE, [N]EXT, [P]REVIOUS, [S]EARCH, OR [Q]UIT."
2160  INPUT A$@ IF A$="" THEN 2150 ELSE A$=UPC$ (A$) @ A$=A$[1,1]
2170  IF A$="Q" THEN RETURN
2180  IF A$="S" THEN QUERY_DB
2190 ON 1+(A$="U")+(A$="D")*2+(A$="N")*3+(A$="P")*4+(A$="S")*5 GOSUB INVAL ,UPDA
TE ,DB_DELETE ,GET_NEXT ,GET_PREV
2200 IF NOT HAVE_RECORD THEN RETURN ELSE FOUND_IT ! end of QUERY
2220 INVAL: DISP "INVALID RESPONSE. PLEASE RE-ENTER." @ RETURN
2230 ! individual QUERY subroutines follow
2240 UPDATE:
2250 GOSUB CHANGE_FIELD
2260 IF NOT CHANGED THEN RECORD$=OLD_RECORD$ @ RETURN
2270 DISP "----- UPDATING THE FILES -----"
2280 !  update key files if key values changed
2290   FOR K=1 TO NUM_KEYS
2300    NEWKEY$,KEY$(K)=RECORD$[F_BEG(KEYFIELD(K)),F_END(KEYFIELD(K))]
2310    IF KEY$(K)=OLD_KEY$(K) THEN NEXT_KEY ! key not changed
2320    SAVEDKEY$=OLD_KEY$(K)
2330    DRN=RN ! in DELETE, RN is updated so the copy should be used
2340    DELETE_KEY S,K,DRN,OLD_KEY$(K)
2350    IF S=137 THEN GOSUB UPDATE_DELETE_ERROR  @ GOTO 2330
2360    CREATE_KEY S,K,RN,KEY$(K)
```

```
2370    IF S=137 THEN GOSUB CREATE_ERROR  @ GOTO 2360
2380 NEXT_KEY: NEXT K
2390   BUFNUM=2 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
2400   PRINT# BUFNUM,RN ; RECORD$ ! update data file
2410   OFF ERROR @ IF RW_ERRFLAG THEN 2390
2420  SAVEDKEY$=RECORD$[F_BEG(KEYFIELD(KY)),F_END(KEYFIELD(KY))]
2430 SEEK_NEXT_KEY S,KY,RN,KEY$(KY) ! get next record after the updated one
2440 IF S=137 THEN GOSUB NEXTSEEK_ERROR  @ GOTO 2430
2450 IF S THEN HAVE_RECORD=0 @ DISP "THERE IS NO RECORD AFTER THE UPDATED RECORD
."
2460 RETURN ! end of UPDATE
2480 DB_DELETE: ! delete the current record
2490  DISP "ARE YOU SURE YOU WANT TO DELETE THIS RECORD? ENTER [YES] OR [NO]."
2500  INPUT A$@ A$=UPC$ (A$) @ IF A$#"YES" THEN RETURN
2510  DISP "----- DELETING THE KEYS AND RECORD FROM FILES -----"
2520  FOR K=1 TO NUM_KEYS
2530   SAVEDKEY$=KEY$(K)
2540   DRN=RN @ DELETE_KEY S,K,DRN,KEY$(K)
2550   IF S=137 THEN GOSUB DELETE_ERROR  @ GOTO 2540
2560  NEXT K
2570  RECORD_MAP$[RN,RN]=" " ! RN is marked empty
2580  NUM_USED=NUM_USED-1
2590  BUFNUM=1 @ RW_ERRFLAG=0 @ ON ERROR GOSUB BUF_ERROR
2600  PRINT# BUFNUM,1 ; NUM_USED,RECORD_MAP$ ! update vital statistics
2610  OFF ERROR @ IF RW_ERRFLAG THEN 2590
2620  SAVEDKEY$=RECORD$[F_BEG(KEYFIELD(KY)),F_END(KEYFIELD(KY))]
2630  SEEK_NEXT_KEY S,KY,RN,KEY$(KY) ! get the record after the deleted one
2640  IF S=137 THEN GOSUB NEXTSEEK_ERROR  @ GOTO 2630
2650  IF S THEN HAVE_RECORD=0 @ DISP "THERE IS NO RECORD AFTER THE DELETED RECOR
D."
2660  RETURN ! end of DB_DELETE
2680 GET_NEXT: ! get the record that comes after the current one in key order
2690  SAVEDKEY$=RECORD$[F_BEG(KEYFIELD(KY)),F_END(KEYFIELD(KY))]
2700  SEEK_NEXT_KEY S,KY,RN,KEY$(KY)
2710  IF S=137 THEN GOSUB NEXTSEEK_ERROR  @ GOTO 2700
2720  IF S THEN HAVE_RECORD=0 @ DISP "END OF KEY FILE REACHED."
2730 RETURN ! end of GET_NEXT
2750 GET_PREV:
2760  SAVEDKEY$=RECORD$[F_BEG(KEYFIELD(KY)),F_END(KEYFIELD(KY))]
2770  SEEK_PRIOR_KEY S,KY,RN,KEY$(KY)
2780  IF S=137 THEN GOSUB PRIORSEEK_ERROR  @ GOTO 2770
2790  IF S THEN HAVE_RECORD=0 @ DISP "END OF KEY FILE REACHED."
2800 RETURN ! end of GET_PREV
2820 DISPLAY_RECORD: CLEAR
2830  DISP "-----------------------------------------------------------------"
2840  DISP TAB (1);"FIELD#";TAB (8);"FIELD NAME";TAB (40);"FIELD VALUE"
2850  DISP "-----------------------------------------------------------------"
2860 FOR F=1 TO NUM_FIELDS
2870  IF ADDING THEN 2890
2880  IF F=KEYFIELD(KY) THEN DISP TAB (1);"-->";! point to the key field
2890  DISP TAB (4);F;TAB (7);FIELD_NAME$(F);TAB (20);RECORD$[F_BEG(F),F_END(F)]
2900 NEXT F
2910  DISP "-----------------------------------------------------------------"
2920 RETURN
2940 SAVE_OLD_RECORD: ! provision for undoing UPDATE or DELETE
2950 ! extract key fields for update and delete
2960   FOR K=1 TO NUM_KEYS
2970     KEY$(K),OLD_KEY$(K)=RECORD$[F_BEG(KEYFIELD(K)),F_END(KEYFIELD(K))]
2980   NEXT K
2990   OLD_RECORD$=RECORD$ @ OLD_RN=RN
3000 RETURN
3020 GET_A_FIELD: ! accept the value of field # F and stuff it in long string
3030   DISP "TYPE THE VALUE OF FIELD # ";F;" [";FIELD_NAME$(F);"]."
3040   INPUT INPBUF$
3050   IF KEY_MAP$[F,F]="Y" THEN INPBUF$=UPC$ (INPBUF$) ! if key capitalize it
3060   IF LEN (INPBUF$)>FIELD_LEN(F) THEN INPBUF$=INPBUF$[1,FIELD_LEN(F)]
3070   RECORD$[F_BEG(F),F_END(F)]=INPBUF$
3080 RETURN
3100 CHANGE_FIELD: OLD_RECORD$=RECORD$ @ CHANGED=0
```

```
3110   DISP "IF YOU WANT TO CHANGE DATA, TYPE THE FIELD NUMBER (1 THROUGH ";NUM_
FIELDS;")."
3120   DISP "ENTER [E]ND WHEN DONE OR [U]NDO THE CHANGES YOU MADE."
3130   INPUT A$@ A$=UPC$ (A$)
3140   IF POS (A$,"E")=1 AND NOT CHANGED THEN RETURN
3150   IF POS (A$,"U")=1 THEN RECORD$=OLD_RECORD$ @ CHANGED=0 @ RETURN
3160   IF POS (A$,"E")=1 THEN GOSUB DISPLAY_RECORD  @ GOTO CHECK_IF_OK
3170   ON ERROR GOTO BAD_NUMBER
3180   F=VAL (A$)
3190   OFF ERROR @ IF F<0 OR F>NUM_FIELDS THEN BAD_NUMBER
3200   GOSUB GET_A_FIELD  @ CHANGED=1
3210   GOTO 3110
3220 BAD_NUMBER: OFF ERROR @ DISP "PLEASE INPUT A VALID RESPONSE." @ GOTO 3110
3230 CHECK_IF_OK: DISP "IS THIS OK? ENTER [Y] OR [N]."
3240   INPUT INPBUF$@ INPBUF$=UPC$ (INPBUF$)
3250   IF POS (INPBUF$,"Y")=1 THEN RETURN ELSE 3110
3260 OPEN_HEADER:
3270 DISP "PLEASE MAKE SURE THE DISC WITH FILE [";DBNAME$;"] IS IN THE DEFAULT D
RIVE."
3280 DISP "PRESS [CONT] KEY WHEN YOU ARE READY TO START." @ PAUSE
3290 DISP "OPENING DATA BASE [";DBNAME$;"]."
3300 ON ERROR GOTO CHECK_FILE
3310 ASSIGN# 1 TO  DBNAME$&"_H"
3320 OFF ERROR
3330  FILE_OPENED=1 ! set file open flag
3340  READ# 1,1 ; NUM_USED,RECORD_MAP$
3350  READ# 1,2 ; MAX_RECORDS,RECORD_LEN,NUM_KEYS,KEY_MAP$,KEYFIELD(),NUM_FIELDS

3360  READ# 1,3 ! move the file pointer
3370  FOR I=1 TO NUM_FIELDS @ READ# 1 ; FIELD_NAME$(I)@ NEXT I
3380  FOR I=1 TO NUM_FIELDS @ READ# 1 ; FIELD_LEN(I),F_BEG(I),F_END(I)@ NEXT I
3390 RETURN
3400 CHECK_FILE: OFF ERROR @ FILE_OPENED=0
3410  DISP "THERE IS A FILE ERROR.  CAN'T OPEN THE FILE."
3420  DISP "IS THE FILE NAME [";DBNAME$;"] CORRECT? ENTER [Y] OR [N]."
3430  INPUT A$@ A$=UPC$ (A$)
3440  IF POS (A$,"Y")=1 THEN OPEN_HEADER ELSE RETURN
3460 CLOSE_FILES: IF NOT FILE_OPENED THEN RETURN
3470  DISP "----- CLOSING FILES -----"
3480  FOR K=1 TO NUM_KEYS
3490   CLOSE_KEY_FILE S,K
3500  NEXT K
3510  ON ERROR GOTO 3540
3520  ASSIGN# 1 TO  *
3530  ASSIGN# 2 TO  * ! deallocate and flush buffers
3540  OFF ERROR @ FILE_OPENED=0 ! reset open flag to closed
3550 RETURN
3570 SHOW_DEFINITION:
3580 !   display the definition of the data base
3590 CLEAR @ DISP "DATA BASE [";DBNAME$;"] DEFINITION SUMMARY"
3600 DISP "MAXIMUM CAPACITY = ";MAX_RECORDS;"  NUMBER OF FIELDS =";NUM_FIELDS
3610 DISP "NUMBER OF ENTRIES IN THE DATA BASE = ";NUM_USED
3620 DISP "-------------------------------------------------------------------"
3630 DISP TAB (2);"FIELD #";TAB (15);"FIELD NAME";TAB (30);"FIELD LENGTH";TAB (5
0);"KEY OR NON-KEY"
3640 FOR I=1 TO NUM_FIELDS
3650 DISP TAB (6);I;TAB (16);FIELD_NAME$(I);TAB (35);FIELD_LEN(I);
3660 DISP TAB (55);KEY_MAP$[I,I]
3670 NEXT I
3680 DISP "-------------------------------------------------------------------"
3690 RETURN
3710 BUF_ERROR: RW_ERRFLAG=1 ! header or data file access error
3720   IF BUFNUM=1 THEN ERRFILE$=DBNAME$&"_H" ELSE ERRFILE$=DBNAME$
3730   OFF ERROR
3740   DISP "ERROR IN ACCESSING FILE ["&ERRFILE$;"]. PLEASE MAKE SURE THE"
3750   DISP "DATA BASE IS IN THE DEFAULT DRIVE. PRESS [END LINE] WHEN DONE."
3760   INPUT A$@ CLEAR
3770   ON ERROR GOTO 3730
3780   ASSIGN# BUFNUM TO  ERRFILE$
3790   OFF ERROR
```

```
3800 RETURN
3820 EXTEND_RECOVER: ! extend a data base or recover a corrupt data base
3830   DIM NEWRECORD_MAP$[100]
3840   GOSUB OPEN_HEADER
3850   IF NOT FILE_OPENED THEN RETURN ! unsuccessful header read
3860   OLDMAX_RECORDS=MAX_RECORDS ! save old file parameters
3870   OLDNUM_USED=NUM_USED
3880   GOSUB SHOW_DEFINITION
3890   IF NOT EXTENDING THEN 4020
3900   DISP "ENTER THE NEW MAXIMUM CAPACITY OF THE DATA BASE. MUST BE <=";MAXDBS
IZE;"."
3910   INPUT NEWMAX_RECORDS
3920   IF NEWMAX_RECORDS>MAXDBSIZE THEN DISP "TOO LARGE. RE-ENTER." @ GOTO 3900
3930   IF NEWMAX_RECORDS<MAX_RECORDS THEN DISP "MUST BE >=";MAX_RECORDS;"." @ GO
TO 3900
3940   DISP "NEW CAPACITY = ";NEWMAX_RECORDS;"."
3950   DISP "IS THIS OK? ENTER [Y]ES, [N]O, OR [C]ANCEL EXTENSION."
3960   INPUT A$@ A$=UPC$ (A$[1,1])
3970   IF A$="C" THEN FILE_OPENED=0 @ RETURN
3980   IF A$="N" THEN 3890
3990   IF A$#"Y" THEN DISP "INVALID RESPONSE." @ GOTO 3950
4000   NEWRECORD_MAP$="" @ NEWRECORD_MAP$[NEWMAX_RECORDS,NEWMAX_RECORDS]=" "
4010   NEWRECORD_MAP$[1,NEWMAX_RECORDS]=RECORD_MAP$
4020   DISP "ENTER THE NEW MASS STORAGE ADDRESS, [701] FOR EXAMPLE, WHERE YOU WA
NT THE NEW FILE"
4030   DISP "CREATED (IF DIFFERENT FROM THE DEFAULT DEVICE) OR PRESS [END LINE].
"
4040   INPUT INPBUF$
4050   IF INPBUF$="" THEN NEWMSTOR=0 @ GOTO 4100
4060   ON ERROR GOTO 4090
4070   NEWMSTOR=VAL (INPBUF$)
4080   OFF ERROR @ GOTO 4100
4090   OFF ERROR @ DISP "INVALID INPUT. PLEASE TYPE A NUMBER." @ GOTO 4020
4100 IF NOT NEWMSTOR THEN OLD_HEADER
4110   DISP "PLEASE INSERT A NEW DISC IN THE DISC DRIVE :D"&VAL$ (NEWMSTOR)&" AN
D LEAVE THE OLD FILE IN"
4120   DISP "THE DEFAULT DRIVE.  PRESS [CONT] WHEN YOU ARE READY." @ PAUSE
4130   ON ERROR GOTO 4160
4140   CREATE DBNAME$&"_H:D"&VAL$ (NEWMSTOR),4
4150   OFF ERROR @ DISP "----- NEW HEADER FILE CREATED -----" @ GOTO 4170
4160   OFF ERROR @ DISP "CAN'T CREATE THE NEW HEADER FILE. PLEASE RETRY." @ GOTO
 4020
4170   NEWRECORD_MAP$="" @ NEWRECORD_MAP$[NEWMAX_RECORDS,NEWMAX_RECORDS]=" "
4180   NEWRECORD_MAP$[1,NEWMAX_RECORDS]=RECORD_MAP$
4190 ASSIGN# 3 TO  DBNAME$&"_H:D"&VAL$ (NEWMSTOR)
4200   PRINT# 3,1 ; NUM_USED,NEWRECORD_MAP$
4210   PRINT# 3,2 ; NEWMAX_RECORDS,RECORD_LEN,NUM_KEYS,KEY_MAP$,KEYFIELD(),NUM_F
IELDS
4220   PRINT# 3,3
4230   FOR I=1 TO NUM_FIELDS @ PRINT# 3 ; FIELD_NAME$(I) @ NEXT I
4240   FOR I=1 TO NUM_FIELDS @ PRINT# 3 ; FIELD_LEN(I),F_BEG(I),F_END(I) @ NEXT
I
4250 ASSIGN# 3 TO  *
4260 GOTO 4330
4280 OLD_HEADER: ! update old header file to reflect extension
4290   IF NOT EXTENDING THEN PURGE_CORRUPT_KEYFILES
4300   PRINT# 1,1 ; NUM_USED,NEWRECORD_MAP$
4310   ASSIGN# 1 TO  *
4330 IF EXTENDING THEN RENAME_OLDKEYFILES
4340 ! else purge corrupt key files
4350 PURGE_CORRUPT_KEYFILES: DISP "----- PURGING CORRUPT KEY FILES -----"
4360    FOR K=1 TO NUM_KEYS
4370      KILL_KEY_FILE S,DBNAME$&"_"&VAL$ (K)
4380    NEXT K
4390    DISP "----- PACKING THE DISC AFTER PURGING KEY FILES -----"
4400    PACK
4410   GOTO MAKE_NEWKEYFILES
4430 RENAME_OLDKEYFILES: IF NEWMSTOR THEN MAKE_NEWKEYFILES
4440   DISP "----- RENAMING THE OLD KEY FILES -----"
4450   FOR K=1 TO NUM_KEYS
```

```
4460       DISP DBNAME$&"_"&VAL$ (K)&"--->"&DBNAME$&"_"&CHR$ (64+K)
4470       RENAME DBNAME$&"_"&VAL$ (K) TO  DBNAME$&"_"&CHR$ (64+K)
4480    NEXT K
4490 !    rename key files from filename_1 to filename_A
4500 MAKE_NEWKEYFILES: IF NEWMSTOR THEN MS$=":D"&VAL$ (NEWMSTOR) ELSE MS$=""
4510    IF EXTENDING THEN MAX_RECORDS=NEWMAX_RECORDS
4520    IF NOT NEWMSTOR THEN 4540
4530    DISP "----- PACKING DISC "&MS$&" ----- " @ PACK MS$
4540    DISP "----- CREATING NEW KEY FILES -----"
4550    FOR J=1 TO NUM_KEYS
4560     KEYFILESIZE=MAX_RECORDS DIV (.8*(253/(FIELD_LEN(KEYFIELD(J))+2)+1)-1)+13

4570     MAKE_KEY_FILE S,DBNAME$&"_"&VAL$ (J)&MS$,FIELD_LEN(KEYFIELD(J)),KEYFILES
IZE
4580     IF S THEN DISP "MAKE_KEY_FILE ERROR. MUST EXIT." @ GOTO MKF_ERR
4590    NEXT J
4600    DISP "----- OPENING NEW KEY FILES -----"
4610    FOR K=1 TO NUM_KEYS
4620     OPEN_KEY_FILE S,DBNAME$&"_"&VAL$ (K)&MS$,K
4630     IF S THEN DISP "OPEN_KEY_FILE ERROR. MUST EXIT." @ END
4640    NEXT K
4650 IF NOT EXTENDING THEN ASSIGN# 2 TO  DBNAME$ @ GOTO RECREATE_KEY_FILES
4660 ! else extending the data file so must create a new extended data file
4670 IF NEWMSTOR THEN CREATE_NEW_FILE
4680 ! creating a new file on the same disc as the old file (must rename)
4690     RENAME DBNAME$ TO  DBNAME$&"_z"
4700 CREATE_NEW_FILE: ! an extended file is created, not executed in recovery
4710    CLEAR
4720    DISP "----- CREATING A NEW DATA FILE ----- "
4730    CREATE DBNAME$&MS$,MAX_RECORDS,RECORD_LEN+3
4740    IF NEWMSTOR THEN ASSIGN# 2 TO  DBNAME$ ELSE ASSIGN# 2 TO  DBNAME$&"_z"
4750                     ASSIGN# 4 TO  DBNAME$&MS$ ! extended data file
4760 ! reconstruct key files by extracting key fields from each record and
4770 ! inserting the keys and corresponding record numbers into key files
4780 RECREATE_KEY_FILES: REC_COUNT=0
4790 IF NOT EXTENDING THEN DISP "----- RESTORING THE KEY FILES -----" @ GOTO 481
0
4800    DISP "--- COPYING RECORDS TO THE NEW FILE AND ADDING KEYS TO THE NEW KEY
FILES ---"
4810   FOR NUM_COPIED=1 TO NUM_USED
4820    REC_COUNT=REC_COUNT+1 @ IF REC_COUNT>MAX_RECORDS THEN 4890 ! can't occur
4830    IF RECORD_MAP$[REC_COUNT,REC_COUNT]=" " THEN 4820 ! RN=REC_COUNT unused
4840    RN=REC_COUNT
4850    READ# 2,RN ; RECORD$
4860 ! if extending data file, copy records from the old file to the new file
4870    IF EXTENDING THEN PRINT# 4,RN ; RECORD$
4880 ! extract the values of key fields and insert them in the new key files
4890    FOR K=1 TO NUM_KEYS
4900     KEY$(K)=RECORD$[F_BEG(KEYFIELD(K)),F_END(KEYFIELD(K))]
4910     CREATE_KEY S,K,RN,KEY$(K)
4920     IF NOT (S=0 OR S=103) THEN DISP "CREATE_KEY ERROR. MUST EXIT." @ END
4930    NEXT K
4940   NEXT NUM_COPIED
4950 IF EXTENDING THEN DISP "----- EXTENSION SUCCESSFUL -----" ELSE DISP "-----
RECOVERY SUCCESSFUL -----"
4960 ! flush and close data file buffers
4970   ASSIGN# 2 TO  *
4980   IF EXTENDING THEN ASSIGN# 4 TO  *
4990   DISP "----- CLOSING KEY FILES ----- "
5000   FOR K=1 TO NUM_KEYS
5010     CLOSE_KEY_FILE S,K
5020   NEXT K
5030 IF NOT EXTENDING OR NEWMSTOR THEN RETURN
5040 ! if extending the old files on the same disc then purge old files
5050    PURGE_ERR=0
5060    DISP "--- PURGING OLD FILES FROM THE DISC IN THE DEFAULT DRIVE ---"
5070    ON ERROR GOTO 5100
5080      PURGE DBNAME$&"_z"
5090    OFF ERROR @ GOTO 5120 ! data file purged
5100    OFF ERROR @ PURGE_ERR=1
```

```
5110   DISP "CANNOT DELETE OLD DATA FILE NAMED ";DBNAME$&"_z."
5120   DISP "----- PURGING OLD KEY FILES -----"
5130   FOR K=1 TO NUM_KEYS
5140     KILL_KEY_FILE S,DBNAME$&" "&CHR$ (64+K)
5150     IF S THEN PURGE_ERR=1 @ DISP "CAN'T PURGE KEY FILE - ";DBNAME$&"_"&CHR$
 (64+K);"."
5160   NEXT K
5170 IF NOT PURGE_ERR THEN DISP "----- OLD FILES SUCCESSFULLY PURGED -----" @ RE
TURN
5180 ! else old files remain unpurged
5190 DISP "PLEASE PURGE THE REMAINING OLD FILES LISTED ABOVE FROM THE DEFAULT DR
IVE."
5200 DISP "YOU CAN PURGE THOSE FILES BY ENTERING [PURGE] filename IN CALCULATOR
MODE."
5210 DISP "THIS PROGRAM IS PAUSING NOW.  PLEASE PURGE THOSE FILES."
5220 DISP "PRESS [CONT] OR [RUN] TO EITHER RESUME OR RERUN THE PROGRAM."
5230 PAUSE
5240 RETURN ! end of EXTEND_RECOVER
5260 FIRSTSEEK_ERROR: DISP "DISC ERROR IN SEEK_FIRST." @ GOTO 5280
5270 ENDSEEK_ERROR: DISP "DISC ERROR IN SEEK_END."
5280   ERRTYPE=1 @ GOTO 5430
5290 NEXTSEEK_ERROR: DISP "DISC ERROR IN SEEK_NEXT_KEY." @ GOTO 5310
5300 PRIORSEEK_ERROR: DISP "DISC ERROR IN SEEK_PRIOR_KEY."
5310   ERRTYPE=2 @ GOTO 5430
5320 SEEK_ERROR: DISP "DISC ERROR IN SEEK_KEY."
5330   ERRTYPE=3 @ GOTO 5430
5340 CREATE_ERROR: DISP "DISC ERROR IN CREATE_KEY." @ ERRTYPE=4 @ GOTO 5380
5350 UPDATE_DELETE_ERROR: ERRTYPE=5 @ GOTO 5370
5360 DELETE_ERROR: ERRTYPE=6
5370   DISP "DISC ERROR IN DELETE_KEY."
5380 DISP "YOUR KEY FILE MAY BE CORRUPT.  DO YOU WANT TO CONTINUE OR EXIT AND"
5390 DISP "RUN RECOVERY ON THE DATA BASE.  ENTER [C]ONTINUE OR [E]XIT."
5400 INPUT A$@ A$=UPC$ (A$[1,1])
5410 IF A$="E" THEN DISP "EXITING THE PROGRAM DUE TO DISC ERROR." @ END
5420 IF A$#"C" THEN DISP "PLEASE ENTER [C]ONTINUE OR [E]XIT." @ GOTO 5400
5430 DISP "PLEASE MAKE SURE ALL FILES ARE ON THE DEFAULT DISC AND THE DRIVE"
5440 DISP "DOOR IS CLOSED.  PRESS [END LINE] WHEN YOU ARE READY."
5450 INPUT A$@ CLEAR
5460   SET_UP S @ ERRFLAG=0
5470 DISP "----- RE-OPENING THE KEY FILES -----"
5480   FOR KEYFILES=1 TO NUM_KEYS
5490     OPEN_KEY_FILE S,DBNAME$&"_"&VAL$ (KEYFILES),KEYFILES
5500     IF NOT (S=0 OR S=103) THEN KEYFILES=13 @ ERRFLAG=1
5510   NEXT KEYFILES
5520 IF NOT ERRFLAG THEN DISP "----- KEY FILES REOPENED -----" @ GOTO 5540
5530   DISP "ERROR IN OPENING KEY FILES." @ BEEP @ WAIT 3000 @ GOTO 5430
5540  CLEAR
5550  ON ERRTYPE GOTO 5570,5590,5640,5650,5660,5670
5570   RETURN ! case of SEEK_FIRST or SEEK_END
5590   KEY$(KY)=SAVEDKEY$ ! case of SEEK_NEXT_KEY or SEEK_PRIOR_KEY
5600   SEEK_KEY S,KY,RN,KEY$(KY)
5610   IF NOT S THEN RETURN
5620   DISP "ERROR IN SEEK_KEY." @ BEEP @ WAIT 3000 @ GOTO 5430
5640   KEY$(KY)=SAVEDKEY$ @ RETURN ! case of SEEK_KEY
5650   KEY$(K)=NEWKEY$ @ RETURN ! case of CREATE_KEY
5660   OLD_KEY$(K)=SAVEDKEY$ @ RETURN ! case of delete in update
5670   KEY$(K)=SAVEDKEY$ @ RETURN ! case of delete
```

Index