



DIGITAL  
RESEARCH™

**Access Manager™**  
Productivity Tool

**Programmer's Guide**

For the  
CP/M® Family of Operating Systems

**Access Manager™  
Productivity Tool  
Programmer's Guide  
for the CP/M® Family  
of Operating Systems**

Copyright © 1983

Digital Research  
P.O. Box 579  
160 Central Avenue  
Pacific Grove, CA 93950  
(408) 649-3896  
TWX 910 360 5001

All Rights Reserved

## COPYRIGHT

Copyright © 1983 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California, 93950.

This manual is, however, tutorial in nature. Thus, the reader is granted permission to include the example programs, either in whole or in part, in his own programs.

## DISCLAIMER

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

## TRADEMARKS

CBASIC and CP/M are registered trademarks of Digital Research. Access Manager, CB80, LK80, MP/M, MP/M II, RMAC, PL/I-80, and Pascal/MT+ are trademarks of Digital Research.

The Access Manager Programmer's Guide for the CP/M Family of Operating Systems was prepared using the Digital Research TEX Text Formatter and printed in the United States of America.

```
*****  
* First Edition: June 1982 *  
* Second Edition: November 1982 *  
* Third Edition: March 1983 *  
*****
```

## Foreword

This programmer's guide contains information and instructions for implementing your application programs with Access Manager™ . It is specifically directed at implementations using the CP/M® family of operating systems.

The Access Manager Programmer's Guide was designed and written as the companion manual to the Access Manager Reference Manual. Please note that there are two separate versions of the programmer's guide, an 8080 and an 8086 version.

Section 1 of this manual describes the general guidelines and restrictions you must observe when using Access Manager with a particular operation system. Sections 2, 3, and 4 provide detailed instructions for using and implementing Access Manager with programs written in CBASIC® Compiler (CB80™ ), PL/I-80™ , and Pascal/MT+™ .



# Table of Contents

## 1 Implementation Guidelines

1.1	Main Access Manager Components . . . . .	1-1
1.2	Memory Requirements for Access Manager Code . . . . .	1-3
1.3	Access Manager Design Constraints . . . . .	1-4
1.4	Multiuser Module under MP/M II . . . . .	1-5
1.4.1	Creating Access Manager Queues . . . . .	1-5
1.4.2	Invoking Shared Routines . . . . .	1-6
1.4.3	Cancelling Shared Routines . . . . .	1-7
1.4.4	Creating Custom Background Servers . . . . .	1-7
1.4.5	Data and Index Files . . . . .	1-8
1.5	Configuring the Single-user Buffer Area . . . . .	1-8

## 2 Using Access Manager with CB80 Applications

2.1	Linking Access Manager to Your Application Program	2-1
2.1.1	Linking Single-user CB80 Applications . . . . .	2-1
2.1.2	Linking Multiuser CB80 Applications . . . . .	2-1
2.2	External Declaration of Access Manager Routines . . . . .	2-2
2.3	Coding Numeric Key Values . . . . .	2-2
2.4	Using the RECREATE.BAS Utility Program . . . . .	2-2
2.5	CB80 Data File Example Listing . . . . .	2-4
2.6	CB80 DATABASE Source Code . . . . .	2-7
2.7	CB80 Source Code Data File Routines . . . . .	2-24
2.7.1	OPEN.DATA.FILE%(DATA.FILE\$,FILE.NO%,RECORD	2-24
2.7.2	CLOSE.DATA.FILE%(FILE.NO%) . . . . .	2-24
2.7.3	NEW.DATA%(FILE.NO%) . . . . .	2-24
2.7.4	RETURN.DATA%(FILE.NO%,DATA.RECORD%,MESSAGE\$)	2-25
2.7.5	DATA.FILE.SIZE%(FILE.NO%) . . . . .	2-26
2.7.6	DATA.FILE.UTILIZATION%(FILE.NO%) . . . . .	2-26

## 3 Using Access Manager with PL/I-80 Applications

3.1	Linking Access Manager to Your Application Program	3-1
-----	----------------------------------------------------	-----

## Table of Contents (continued)

3.1.1	Linking Single-user PL/I-80 Applications . . .	3-1
3.1.2	Linking Multiuser PL/I-80 Applications . . .	3-1
3.2	External Declaration of Access Manager Routines . .	3-2
3.3	Coding Numeric Key Values . . . . .	3-3
3.4	Using the RECREATE.PLI Utility Program . . . . .	3-4
3.5	PL/I-80 Data File Example . . . . .	3-5
3.6	PL/I-80 DATABASE Source Code . . . . .	3-9
<b>4</b>	<b>Using Access Manager with Pascal/MT+ Applications</b>	
4.1	Linking Access Manager to Your Application Program	4-1
4.1.1	Linking Single-user Pascal/MT+ Applications	4-1
4.1.2	Linking Multiuser Pascal/MT+ Applications .	4-2
4.2	External Declaration of Access Manager Routines . .	4-2
4.3	Coding Numeric Key Values . . . . .	4-2
4.4	Using the RECREATE.SRC Utility Program . . . . .	4-3
4.5	Pascal/MT+ Data File Example . . . . .	4-5
4.6	Pascal/MT+ DATABASE Source Code . . . . .	4-9

# Tables and Listings

## Tables

1-1.	Access Manager Code Requirements . . . . .	1-4
1-2.	CP/M and MP/M II Design Constraints . . . . .	1-5
1-3.	Queue Space Requirements . . . . .	1-6
1-4.	Suggested Index File Record Lengths . . . . .	1-8
2-1.	Example CB80 Recreate Parameter File . . . . .	2-3
2-2.	Data Record Field . . . . .	2-25
3-1.	Example PL/I-80 Recreate Parameter File . . . . .	3-5
4-1.	Example Pascal/MT+ Recreate Parameter File . . . . .	4-4

## Listings

2-1.	CB80 Data File Example . . . . .	2-4
2-2.	DATABASE.BAS Source Code . . . . .	2-8
3-1.	PL/I-80 Data File Example . . . . .	3-6
3-2.	DATABASE.BAS Source Code . . . . .	3-10
4-1.	Pascal/MT+ Data File Example . . . . .	4-5
4-2.	DATABASE.SRC Source Code . . . . .	4-10





# Section 1

## Implementation Guidelines

### 1.1 Main Access Manager Components

Your Access Manager distribution disk contains the following files:

- Single-user subroutine libraries

AM80CB80.IRL	A complete, binary-relocatable, indexed library of index and data file routines for CB80 application programs.
AM80PLI.IRL	A complete, binary-relocatable, indexed library of index and data file routines for PL/I-80 application programs.
AM80PASC.ERL	A complete, binary-relocatable library of index and data file routines for Pascal/MT+ application programs.

- Single-user buffer modules

AM80BUF.IRL	A relocatable object module containing a prespecified buffer area of 3,560 (decimal) bytes for CB80 and PL/I-80 application programs.
AM80BUF.ERL	A relocatable object module containing a prespecified buffer area of 3,560 (decimal) bytes for Pascal/MT+ application programs.
AM80BUF.ASM	The assembly language source code for the buffer modules. Contains entry points (AM8FCB and AM8END) that define the beginning and end of the buffer area.
SETAMBUF.COM	A program to change the buffer module sizes without reassembling AM80BUF.ASM.

- External procedure declarations

AM80EXTR.BAS	External procedure declarations for CB80 application programs.
AM80EXTR.PLI	External procedure declarations for PL/I-80 application programs.

AM80EXTR.PSC External procedure declarations for Pascal/MT+ application programs.

- Multiuser application interfaces

AMQ8CB80.IRL Relocatable interface module to coordinate compiled CB80 application programs and the shared Access Manager routines (AM80.PRL).

AMQ8PLI.IRL Relocatable interface module to coordinate compiled PL/I-80 application programs and the shared Access Manager routines (AM80.PRL).

AMQIPASC.ERL Relocatable interface module to coordinate compiled Pascal/MT+ application programs and the shared Access Manager routines (AM80.PRL).

- Background server routines

AM80.PRL A page relocatable program containing all multiuser routines. AM80.PRL runs in its own memory segment as a shared, background server.

STOPAM80.PRL A page relocatable utility program that closes all open index and data files, terminates AM80.PRL, and releases its memory segment.

AM80x.RSP Resident System Processes to include in the MP/M™ operating system at system generation time (when GENSYS is run). It sets up the interprocess queues for a multiuser system. For example, AM803.RSP reserves queues for up to three users.

- Background server customization routines

The following files contain the code you need to create custom versions of AM80.PRL. Additional instructions can be found under "Creating Custom Background Servers" in Section 1.4.1.

AM80MBUF.ASM Determines the number and size of buffers and reserves the actual space for them. You can significantly affect the size of AM80.PRL by changing this module.

AM80SERV.REL Contains the message handling code that communicates with the multiuser interfaces to coordinate the sharing of Access Manager routines.

- AM80B.IRL**                    Contains the Access Manager routines for a multiuser environment.
- Host language data file routines
- CDATA.BAS**                    These routines are similar to the data file functions described in Section 3 of your Reference Manual, except they are written in CB80 source code. (See Section 2, of the CBASIC Compiler (CB80) Language Programming Guide.)
- Utility programs
- RECREATE**                    A general purpose program for rebuilding index and data files. Source versions of the program are provided in CB80, PL/I-80, and Pascal/MT+. Section 5 of your Reference Manual contains a complete description of the RECREATE utility program.
- DATABASE**                    A complete data base example for single-user or multiuser environments. Source code is provided in CB80, PL/I-80, and Pascal/MT+.

## 1.2 Memory Requirements for Access Manager Code

[SINGLE] The Access Manager modular design ensures that your application program uses only those parts of Access Manager that are actually required. The calls to Access Manager embedded in the application program determine which modules the linking loader brings into the final file.

[MULTI] Only a small interface module combines with your application program. The actual Access Manager code resides in a separate memory segment.

Table 1-1 shows the Access Manager code requirements for single-user and multiuser environments. The following standard abbreviations are used: B means bytes and K means kilobytes.

**Table 1-1. Access Manager Code Requirements**

Module	Memory in Bytes	
	SINGLE	MULTI
Kernel: system initialization and index file setup and searching	5.3K	---
Key value insertion	1.5K	---
Key value deletion	1.8K	---
Data file routines	2.6K	---
Buffer Area	0.6K+	---
Background server (w/o buffer area)	---	16.4K
Minimal buffer area	---	4.5K
Application program interface	---	2.0K
Queue space requirements:		
for 3 users	---	768B
for 4 users	---	1024B
for 6 users	---	1280B
for 8 users	---	1792B

### 1.3 Access Manager Design Constraints

When using Access Manager, the following design constraints apply in both single-user and multiuser environments:

- Data records in a data file must all be the same length.
- Data records must be a minimum of four bytes in length.
- Access Manager reserves the first 128 bytes of every data file for recording status information.
- Key values must not exceed a length of 48 bytes.
- Data record numbers (pointers) associated with key values must not exceed a length of four bytes.
- The length of an index file record must be a multiple of 128 bytes; for example, 128, 256, 512, 1024, etc.
- There must be a minimum of four key values in any given index file record.
- A minimum of three buffers must be allocated for Access Manager.

If your application program runs under CP/M or MP/M II™, observe the design constraints shown in Table 1-2. In the table megabyte is abbreviated as MB.

**Table 1-2. CP/M and MP/M II Design Constraints**

Design Constraint	CP/M	MP/M II
Maximum number of key values per index file record	124	124
Maximum index file size	8 MB	8 MB
Maximum number of index files that can be open at a given time	10	40
Maximum data file size	8 MB	32 MB
Maximum number of data files that can be open at a given time	20	40
Maximum number of Access Manager disk buffers	18	60
Maximum number of users in a multiuser environment	N/A	8

#### 1.4 Multiuser Module under MP/M II

Correct implementation of Access Manager under MP/M II requires proper use of the following components:

AM80x.RSP creates the Access Manager queues.

AM80.PRL contains the shared index file, data file, and buffer area.

##### 1.4.1 Creating Access Manager Queues

The queues created by AM80x.RSP (where "x" = 3, 4, 6, or 8) coordinate the multiuser keyed file accessing requests. This queue space must be reserved in common memory, meaning memory that is accessible to all users. Table 1-3 shows the queue space memory requirements.

Table 1-3. Queue Space Requirements

Maximum Number of Users	Module Name	Memory in Bytes
3	AM803.RSP	768
4	AM804.RSP	1024
6	AM806.RSP	1280
8	AM808.RSP	1792

For a given number of users, the task of reserving queue space need only be performed once during MP/M system generation. The MP/M utility program GENSYS.COM automatically prompts you to determine if a Resident System Process (any file with a .RSP extension) should be included in the operating system. To set up a three-user system for Access Manager, respond with Y, for Yes, when GENSYS prompts you about AM803.RSP. Respond with N, for No, when prompted about the other AM80x.RSP files. GENSYS prompts you about an RSP file only if such a file is on the GENSYS disk.

After you complete the GENSYS procedure, a new MP/M system is written to a disk file (MPM.SYS). The next time you boot the system, the new version of MP/M resides on disk.

If there is not enough common memory for the Access Manager queue space and any other optional components of your operating system, GENSYS tells you and requests a retry.

#### 1.4.2 Invoking Shared Routines

AM80.PRL contains the shared Access Manager code designed to run in its own memory segment under MP/M. As distributed, it requires less than 32K bytes of memory (although the AM80.PRL file is somewhat larger because it contains a byte relocation map), supports up to forty index files and forty data files, and uses twenty buffers with a node size of 512 bytes (NNSEC%=4). Instructions for changing this configuration can be found under "Creating Custom Background Servers" in Section 1.4.4.

To start AM80.PRL under MP/M, type

```
AM80
```

If Access Manager starts successfully, a message similar to the following appears on your screen:

```
-----
ACCESS MANAGER(tm) 8080                               Version 1.1
Serial No. AM-9999-000000                             All Rights Reserved
Copyright (c) 1982,1983                               Digital Research, Inc.
-----
```

Access Manager(tm) is ready for y users.

Access Manager then detaches from the console and AM80.PRL is waiting in the background to service multiuser application programs.

If Access Manager encounters a problem at start-up, one of the following messages appear on the console:

Access Manager could not open queues. Has an Access Manager RSP been included at GENSYST?

Access Manager Background Server (AM80.PRL) has illegal SETUP parameters. Check AM80MBUF.ASM for proper setup.

Access Manager could not open lock file. Are the disk and/or directory full?

Access Manager could not initialize lock file. Call Digital Research.

### 1.4.3 Cancelling Shared Routines

To close all open index and data files and free the memory segment occupied by AM80.PRL, type the command:

```
STOPAM80
```

If successful, the following message appears on the console:

```
Access Manager (tm) Terminated
```

If AM80.PRL is not running when you attempt to start STOPAM80, this message appears on the console:

Be sure that Access Manager is operational. Run MPMSTAT to see.

### 1.4.4 Creating Custom Background Servers

There are two parameters affecting the size of AM80.PRL that can be modified. These parameters are defined by DW statements in the AM80MBUF.ASM file. The parameters are the following:

- NBUFS%      The number of index file I/O buffers. NBUFS% must be set to at least three. However, 20 is a more realistic value for satisfactory multiuser operation. NBUFS% must not exceed 60.
- NNSEC%      The number of 128-byte sectors comprising each index file node. Must be at least one; four sectors are recommended.



Once you enter the appropriate values for the preceding parameters, you must change the amount of storage reserved by AM80MBUF. The storage is reserved with a DS statement. The amount of storage required is given by this expression:

$$2504 + (\text{NBUFS}\% * ((128 * \text{NNSEC}\%) + 52))$$

After AM80MBUF.ASM is assembled, you can create a new AM80.PRL with the following command line:

```
LINK AM80=AM80SERV[A,OP],AM80MBUF,AM80B.IRL[S]
```

#### 1.4.5 Data and Index Files

When Access Manager runs in a multiuser environment, index and data files are opened in the locked mode. This places total control of these files under AM80.PRL. Further, the files are in the directory of the MP/M user area from which AM80.PRL is started. In most environments, AM80.PRL is started from user area zero.

#### 1.5 Configuring the Single-user Buffer Area

The primary parameters affecting the buffer area size are NNSEC% (the number of sectors per index file record) and NBUFS% (the number of index file buffers). NNSEC% should be set to four if compatibility of your application program with other software is a factor. However, if response time is a more critical issue, refer to Table 1-4 for suggested NNSEC% values.

Note that NNSEC% determines the length of the records in the index file. For example, if NNSEC% is four, the resulting index file record length is 512 bytes, regardless of the physical sector size of the disk.

**Table 1-4. Suggested Index File Record Lengths**

Physical Sector Size of Disk	Suggested Values for NNSEC%*
128	2
256	2,4
512	2,4
1024	4,8

\* NNSEC% specifies the number of 128-byte sectors per index file record.

Within the guidelines of Table 1-4, the selection of a value for NNSEC% is usually based on the length of the keys. Long keys lead to higher values of NNSEC% to reduce the levels of the B-Tree index structure.

For any specified value of NNSEC%, the more buffers (larger NBUFS%) there are, the fewer node accesses you need to retrieve a key value. However, when processing one index file at a time, the payoff from adding buffers diminishes rapidly when five or six buffers are already in use. If more than one index file is in use at a time, increase the number of buffers beyond six. Provided memory space is available, three buffers per index file (in active use at one time) is a reasonable guideline. Buffers are not assigned to individual index files. They are shared according to a least-recently-used priority scheme, which ensures the active index files make full use of the buffers.

The amount of available memory determines the size of the buffer area. If there is very little memory available for buffers, you can reduce NBUFS% to the minimum level of three.

The required buffer size for any given specification of the maximum number of index files (NKEYS%), the node size (NNSEC%), the number of buffers (NBUFS%), and the number of data files (NDATF%), can be calculated like this:

$$((NKEYS\% + NDATF\%) * 44) + (NBUFS\% * ((NNSEC\% * 128) + 52))$$

For example, if NKEYS%=3, NBUFS%=6, NDATF%=1, and NNSEC%=4, a buffer size of 3,560 (decimal) bytes is required. Once this buffer space is reserved, any combination of the four determining parameters that stays within 3,560 bytes can be passed to the SETUP routine, which sets up the way the buffer area is used.

If you have the RMAC™ assembler, which generates relocatable object files (REL files), you can change AM80BUF.ASM, then reassemble it to create a new AM80BUF.REL. Or, you can run the program SETAMBUF.COM, which is on your distribution disk. Just make sure AM80BUF.IRL and AM80BUF.ERL are on the same disk as SETAMBUF.COM. SETAMBUF modifies the buffer modules according to your specifications.

End of Section 1

## Section 2

### Using Access Manager with CBASIC Compiler (CB80) Applications

This section contains instructions for implementing Access Manager with application programs coded in CB80.

There are two examples provided in this section. The first shows the use of many Access Manager functions described in Section 3 of your Reference Manual, and how to use CB80 strings for data file buffer areas. The second example, DATABASE.BAS, illustrates the use of multiple index and data files.

#### 2.1 Linking Access Manager to Your Application Program

This section discusses a CB80 application program call MYPROG that you write and compile to produce a binary relocatable file.

##### 2.1.1 Linking Single-user CB80 Applications

You must link your compiled application program to the appropriate Access Manager subroutine library and index file buffer module. The following command line can be used to create an executable version of MYPROG:

```
LK80 MYPROG,AM80CB80.IRL,AM80BUF.IRL
```

AM80BUF contains the buffer area beginning with entry point AM8FCB and ending with AM8END.

Because AM80CB80 and AM80BUF are indexed, relocatable libraries, LK80™ places them in the root module in case CB80 overlay structures are used. Before linking, be sure AM80BUF is large enough to contain your buffers (as specified in the SETUP function). You can use SETAMBUF to create a correctly sized buffer module.

##### 2.1.2 Linking Multiuser CB80 Applications

If your single-user version of MYPROG is coded with appropriate data locking procedures, you do not have to recompile it to create a multiuser version. All that is necessary is to relink the program.

You must link your compiled application program to the appropriate Access Manager multiuser interface. The interface makes the queue calls to the shared code in the background server. The background server resides in its own memory segment.

To create a COM file that calls the Access Manager background server, use the following command line:

```
LK80 MYPROG,AMQ8CB80.IRL
```

To create a PRL file that calls the Access Manager background server, use LINK.COM (which comes with MP/M II) as follows:

```
LINK MYPROG[OP,A],AMQ8CB80.IRL
```

Note that if you use LINK.COM, your application program cannot use the CB80 overlay facility.

No buffer area module (such as AM80BUF) is permitted in the multiuser link statement. Whereas AM80CB80 contains the actual Access Manager code, AMQ8CB80 simply contains the message handler necessary to get the shared Access Manager code (contained in AM80.PRL) to perform the necessary actions.

## 2.2 External Declaration of Access Manager Routines

CB80 requires that external routines (those not coded in the program module but referenced by it) be explicitly declared. The file AM80EXTR.BAS contains the external function declarations for the entire set of Access Manager functions. Use the %INCLUDE feature of CB80 to make these external declarations a part of your application program.

## 2.3 Coding Numeric Key Values

Refer to the discussion of "Coding Numeric Key Values" under the ADDKEY function description in Section 3 of your Reference Manual.

## 2.4 Using the RECREATE.BAS Utility Program

RECREATE.BAS contains the CB80 source code for the RECREATE utility program. You can change the source code in whatever way you want.

To create RECREATE.COM, compile RECREATE.BAS using CB80 and then link as follows:

```
LK80 RECREATE,AM80CB80.IRL,AM80BUF.IRL
```

The buffer area for RECREATE is 4,600 bytes based on the following parameter values:

- NNSEC% = 4
- NBUFS% = 8
- NDATA% = 1
- NKEYS% = 1

Note that only one data file and one index file are open at the same time RECREATE is running. Use SETAMBUF to configure AM80BUF.IRL.

Table 2-1 shows the layout and content of records in a Recreate Parameter File. This particular example file can be used to reconstruct DATABASE.BAS (see Listing 2-2).

**Table 2-1. Example CB80 Recreate Parameter File**

Record Type	Contents
Header	1,4
Data File	CUSTOMER.DAT,100,3,0
Index File	NAME.IDX,10,0,1,1,Y
Key Part	22,8
Index File	NUMB.IDX,4,0,0,1,N
Key Part	2,4
Index File	ZIPC.IDX,11,0,1,1,Y
Key Part	84,9

If you want to change the capacity of the RECREATE program, and hence its memory requirements, note the following parameters and associated DIMENSION statements:

- **MAX.NO.KEYS%** and **MAX.NO.KEY.PARTS%** specify the maximum number of index files associated with a data file and the maximum number of fields comprising a key value, respectively. If the value for either of these parameters is increased in RECREATE.BAS, the following dimension statements must be modified to reflect the changes:

```
DIM INDEX.NAME$( ...
DIM AUTO.SUFFIX$( ...
```

- **MAX.SORT%** determines the maximum number of key values that will be buffered by RECREATE.BAS before being sorted and added to the index file being recreated. If MAX.SORT% is increased, the following dimension statement must be changed.

```
DIM KEYVAL$( ...
```

The routine SORT.SETUP in RECREATE.BAS uses the FRE and MFRE functions of CB80 to determine the amount of available memory for buffered key values. The actual number of key values buffered is stored in NO.SORT%. For long key lengths, the memory space available limits NO.SORT%. The value for NO.SORT% can be made more conservative by reducing the values of G.SORT and M.SORT before NO.SORT% is computed.

## 2.5 CB80 Data File Example Listing

In the following listing the CB80 SADD function determines the value of the buffer pointer (Access Manager parameter BUFFER%) for the READAT and WRDAT routines. The result of SADD is increased by two because each string variable in CB80 has a two-byte header containing the length of the string.

When Access Manager fills in the input buffer INP.BUFFERS during the READAT function, the two-byte length header is not affected. Therefore, you can use one such string input buffer for all the data files if it is long enough to accommodate the longest record length.

However, the output buffer OUT.BUFFER\$ is constantly adjusted because it is reconstructed for each WRDAT function. Therefore, it is not advisable to use the input buffer for output. Reserve the input buffer for input only; create the output buffer strings as needed.

```
REM -----
REM AM80 External Declarations
REM -----
%INCLUDE AM80EXTR.BAS

REM -----
REM Exception Processing Routines
REM -----

DEF ERROR.HANDLER(LOCALE)
  INTEGER LOCALE

  PRINT "ERROR at ";LOCALE;" with code ";ERRCOD
  STOP
FEND

DEF LOCK.CONFLICT(LOCALE)
  INTEGER LOCALE

  PRINT "LOCK Conflict at ";LOCALE;" with code ";LOKCOD
FEND
```

### Listing 2-1. CB80 Data File Example

```

REM -----
REM Lock Parameter Setup
REM -----
N.LOCK% = 0    REM No lock request
S.LOCK% = 1    REM Shared record lock
X.LOCK% = 2    REM Exclusive record lock
S.FILE% = 3    REM Shared file lock
X.FILE% = 4    REM Exclusive file lock

REM -----
REM System Initialization Parameters
REM -----
NBUF% = 3      REM 3 buffers
NKEYS% = 1     REM 1 index file
NNSEC% = 4     REM 512-byte index file record length
NDATF% = 1     REM 1 data file
ERROPT% = 1    REM Trap user errors
PROGID% = -1   REM Program ID assigned to MP/M console no.
TIMOUT% = 3    REM Background server time-out delay

REM -----
REM Initialize System
REM -----
PROGID% = INTUSR(PROGID%,ERROPT%,TIMOUT%)
IF ERRCOD <> 0 THEN \
    CALL ERROR.HANDLER(1)
IF SETUP(NBUF%,NKEYS%,NNSEC%,NDATF%) <> 0 THEN \
    CALL ERROR.HANDLER(2)

REM -----
REM Open Files
REM -----
FILE.NO% = -1   REM Automatic file number assignment
RECORD.LEN% = 32
FILE.NAME$ = "K:PART.DAT"
FILE.NO% = OPNDAT(FILE.NO%,S.FILE%,FILE.NAME$,RECORD.LEN%)
IF ERRCOD <> 0 THEN \
    CALL ERROR.HANDLER(3)
IF LOKCOD <> 0 THEN \
    CALL LOCK.CONFLICT(3)

REM -----
REM Create input buffer area and buffer pointer. The buffer
REM pointer = SADD + 2 because SADD points to the two-byte
REM length header which precedes the actual string.
REM -----
REM          1          2          3
INP.BUFFER$ = "12345678901234567890123456789012"
BUFFER.PTR% = SADD(INP.BUFFER$) + 2

```

Listing 2-1. (continued)

```
REM -----
REM Set exclusive lock on data record no. 65686
REM -----

DRN2% = 1
CALL SETDAT(DRN2%)          REM Set high-order bytes to 1
                            REM which implies a base of 65536.
DRN% = 150                  REM 65686 = 65536 + 150
IF SETLOK(FILE.NO%,X.LOCK%,DRN%) <> 0 THEN \
    CALL LOCK.CONFLICT(4)

REM -----
REM Read data record
REM -----

CALL SETDAT(DRN2%)
IF READAT(FILE.NO%,DRN%,BUFFER.PTR%) <> 0 THEN \
    CALL ERROR.HANDLER(4)

REM -----
REM Parse buffer into working variables
REM -----

PART.NO$ = LEFT$(INP.BUFFER$,4)
PART.NAME$ = MID$(INP.BUFFER$,5,20)
PART.QUAN = VAL(RIGHT$(INP.BUFFER$,8))

REM -----
REM Update data record
REM -----

PART.QUAN = PART.QUAN - 100.

REM -----
REM Create output buffer
REM -----

OUT.BUFFER$ = PART.NO$ + PART.NAME$ + \
    LEFT$(STR$(PART.QUAN),8)
BUFFER.PTR% = SADD(OUT.BUFFER$) + 2

REM -----
REM Write updated record
REM -----

CALL SETDAT(DRN2%)
IF WRDAT(FILE.NO%,DRN%,BUFFER.PTR%) <> 0 THEN \
    CALL ERROR.HANDLER(5)

REM -----
REM Release record lock
REM -----

CALL SETDAT(DRN2%)
IF FRELOK(FILE.NO%,X.LOCK%,DRN%) <> 0 THEN \
    CALL LOCK.CONFLICT(6)
```

Listing 2-1. (continued)



```
REM -----  
REM Close data file and release file lock  
REM -----  
  
IF CLSDAT(FILE.NO%) <> 0 THEN \  
    CALL ERROR.HANDLER(7)  
IF FRELOK(FILE.NO%,S.FILE%,0) <> 0 THEN \  
    CALL LOCK.CONFLICT(7)  
  
STOP
```

### Listing 2-1. (continued)

## 2.6 CB80 DATABASE Source Code

Your Access Manager distribution disk contains sample code for building and maintaining a data base in CB80. The code is designed so you can add or substitute your own key attributes as required. The sample code is on your distribution disk in a file named DATABASE.BAS.

DATABASE.BAS demonstrates the integration of Access Manager with CB80 applications. It builds a name and address data base and provides facilities for examining, updating, and/or listing the information contained therein. You might also want to use routines from DATABASE.BAS directly in your application programs.

[SINGLE] To create DATABASE.COM, compile DATABASE.BAS with CB80.COM and link as follows:

```
LK80 DATABASE,AM80CB80.IRL,AM80BUF.IRL
```

[MULTI] In the multiuser environment, your link statement should be entered as follows:

```
LK80 DATABASE,AMQ8CB80.IRL
```

Note that Listing 2-2 of DATABASE.BAS might not include recent changes. You should always treat the copy on your distribution disk as the definitive version.

```
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM      DATABASE EXAMPLE   VERSION 1.05   4/17/82   1411
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM      INTERFACE TO AM80 (tm)
REM
REM
REM      AM80EXTR.BAS CONTAINS THE EXTERNAL DEFINITIONS OF THE
REM      AM80 ROUTINES
REM
%INCLUDE AM80EXTR.BAS

REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM      SET-UP DATABASE FIELD & KEY DESCRIPTORS
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
DIM FLD.NAME$(7),FLD.LEN$(7),NEW.FLD$(7),OLD.FLD$(7)
DIM FLD.PTR$(7)
MAX.FIELD% = 7:NO.FIELDS% = MAX.FIELD% + 1
YES% = 1 : NO% = 0
```

**Listing 2-2. DATABASE.BAS Source Code**

```

FLD.NAME$(0) = "Customer Number" :FLD.LEN%(0) = 4
FLD.NAME$(1) = "First Name"      :FLD.LEN%(1) = 16
FLD.NAME$(2) = "Last Name"       :FLD.LEN%(2) = 20
FLD.NAME$(3) = "Street Address"  :FLD.LEN%(3) = 20
FLD.NAME$(4) = "City"           :FLD.LEN%(4) = 20
FLD.NAME$(5) = "State"          :FLD.LEN%(5) = 2
FLD.NAME$(6) = "Zipcode"        :FLD.LEN%(6) = 9
FLD.NAME$(7) = "Customer Status" :FLD.LEN%(7) = 8

DIM KEY.NAME$(2),KEY.LEN%(2),KEY.MAP%(2),KEY.TYPE%(2),KEY.NUM%(2),KEY.DUP%(2)
MAX.KEY% = 2
KEY.LEN%(0)=10:KEY.TYPE%(0)=0:KEY.MAP%(0)=2 REM KEY 0 = LAST NAME
KEY.LEN%(1)=11:KEY.TYPE%(1)=0:KEY.MAP%(1)=6 REM KEY 1 = ZIPCODE
KEY.LEN%(2)=4 :KEY.TYPE%(2)=0:KEY.MAP%(2)=0 REM KEY 2 = CUST NUMBER

UNIQ.KEY% = 2 REM USED IN TEST OF UNIQUENESS

FOR KEY% = 0 TO MAX.KEY%
  IF KEY% = UNIQ.KEY% THEN \
    KEY.DUP%(KEY%) = NO% \
  ELSE \
    KEY.DUP%(KEY%) = YES%
  KEY.NAME$(KEY%) = FLD.NAME$(KEY.MAP%(KEY%))
NEXT KEY%

DIM INDEX.NAME$(2)
INDEX.NAME$(0) = "NAME.IDX"
INDEX.NAME$(1) = "ZIPC.IDX"
INDEX.NAME$(2) = "NUMB.IDX"

NLOCK% = 0 REM IGNORE LOCKS
SLOCK% = 1 REM SHARED RECORD LOCK
XLOCK% = 2 REM EXCLUSIVE RECORD LOCK
SFILE% = 3 REM SHARED FILE LOCK
XFILE% = 4 REM EXCLUSIVE FILE LOCK
RLOCK% = 5 REM RELEASE SLOCK% OR XLOCK%

REM
REM ++++++
REM
REM BEGINNING OF UTILITY FUNCTIONS
REM
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM CLEAR SCREEN ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF CLEAR.SCREEN%
  FOR DUMMY% = 1 TO 24
    PRINT
  NEXT DUMMY%

```

Listing 2-2. (continued)

```

RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          MAIN MENU ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF MAIN.MENU%
PRINT TAB(21);"  AM80(tm) DEMONSTRATION" :PRINT
PRINT TAB(20);"Customer Database Operations"
PRINT TAB(20);"          Terminal ";TERMINAL%
PRINT TAB(20);"*****":PRINT :PRINT
PRINT TAB(5);"1. Enter New Customers"
PRINT TAB(5);"2. Scan/Update/Delete Customer Records"
PRINT TAB(5);"3. List Customer Records"
PRINT TAB(5);"4. Database Statistics"
PRINT TAB(5);"5. Save All Files & Restart Operations"
PRINT TAB(5);"6. Terminate Operations":PRINT :PRINT
1000 INPUT "Enter desired operation number>>";OP%
IF OP%<1 OR OP%>6 THEN PRINT :PRINT :GOTO 1000
MAIN.MENU% = OP%
RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          SELECT SEARCH KEY ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF SEARCH.KEY%
DUMMY% = CLEAR.SCREEN%
PRINT TAB(25);"Customer Database Search Keys":PRINT :PRINT
FOR KEY% = 0 TO MAX.KEY%
  KEY.NO% = KEY% + 1
  PRINT TAB(5);KEY.NO%;"- ";KEY.NAME$(KEY%)
NEXT KEY%
1040 PRINT :PRINT
INPUT "Enter desired key number>>";OP%
IF OP%<1 OR OP%>NO.KEYS% THEN 1040
SEARCH.KEY% = OP%-1
RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          ERROR HANDLING
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF ERROR.TYPE%(TYPE%)
PRINT
PRINT \

```

Listing 2-2. (continued)

```

"User Error #";ERRCOD;" occurred while trying to ";
ON TYPE% GOTO 9210,9230,9250,9290,9300,9320,9330, \
          9340,9350,9360,9370,9380,9390,9400,9410,9420
9210  PRINT "open ";INDEX.NAME$(KEY%) : GOTO 9700
9230  PRINT "search ";KEY.NAME$(KEY%);" Index File" : GOTO 9500
9250  PRINT "save ";INDEX.NAME$(KEY%) : GOTO 9600
9290  PRINT "remove old key from ";INDEX.NAME$(KEY%) : GOTO 9500
9300  PRINT "enter key into ";INDEX.NAME$(KEY%) :GOTO 9500
9320  PRINT "delete key from ";INDEX.NAME$(KEY%) :GOTO 9500
9330  PRINT "save ";FILE.NAME$: :KEY% = -1:GOTO 9600
9340  PRINT "get a new data record";" (";FILE.NO%);" :GOTO 9700
9350  PRINT "delete data record #";DRN% :GOTO 9700
9360  PRINT "open ";FILE.NAME$;" (";FILE.NO%);" :GOTO 9700
9370  PRINT "read data record #";DRN%:GOTO 9700
9380  PRINT "write data record.":GOTO 9700
9390  PRINT "release shared file lock on ";FILE.NAME$:GOTO 9700
9400  PRINT "initialize user.": STOP
9410  PRINT "close ";FILE.NAME$: :KEY% = -1:GOTO 9600
9420  PRINT "close ";INDEX.NAME$(KEY%) : GOTO 9600
9500  CALL CLSDAT(FILE.NO%)
      FOR T.KEY% = 0 TO MAX.KEY%
          IF T.KEY% <> KEY% THEN CALL CLSIDX(KEY.NUM%(T.KEY%))
      NEXT T.KEY%
      GOTO 9700 REM STOP ERROR MESSAGE
9600  T.KEY% = KEY% + 1
      IF T.KEY%>MAX.KEY% THEN STOP
      FOR KEY% = T.KEY% TO MAX.KEY%
          CALL CLSIDX(KEY.NUM%(KEY%))
      NEXT KEY%
9700  PRINT
      PRINT "DEMONSTRATION TERMINATING WITH ERROR CODE #";ERRCOD
      STOP

FEND
DEF LOCK.TYPE%(TYPE%)
  PRINT "Lock Type: ";TYPE%;" Lock Code:";LOKCOD
  CALL CLSDAT(FILE.NO%)
  FOR T.KEY% = 0 TO MAX.KEY%
      CALL CLSIDX(KEY.NUM%(T.KEY%))
  NEXT T.KEY%
  STOP

FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          STRIP TRAILING BLANKS
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF STRIP.BLANKS$(OLD.VAL$,FLD%)
  FOR TEST% = FLD.LEN%(FLD%) TO 1 STEP -1
    IF MID$(OLD.VAL$,TEST%,1) <> " " THEN \
      STRIP.BLANKS$ = LEFT$(OLD.VAL$,TEST%) :\
      RETURN
  NEXT TEST%
  STRIP.BLANKS$ = ""

```

Listing 2-2. (continued)

```

        RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          READ DATA RECORD ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF READ.CUST%(DRN%)
  IF READAT(FILE.NO%,DRN%,INPBUF.PTR%) <> 0 THEN \
    DUMMY% = ERROR.TYPE%(11)
  OFFSET% = 2          REM SKIP DELETE FLAG FIELD
  FOR D.FLD% = 0 TO MAX.FIELD%
    OLD.FLD.VAL$ = MID$(INPBUF$,OFFSET%,FLD.LEN%(D.FLD%))
    OLD.FLD$(D.FLD%) = \
      STRIP.BLANKS$(OLD.FLD.VAL$,D.FLD%)
    OFFSET% = OFFSET% + FLD.LEN%(D.FLD%)
  NEXT D.FLD%
  RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          LIST CUSTOMER RECORD ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF PRINT.CUST%
  IF ROUTE$ = "Y" THEN \
    LPRINTER
  PRINT
  PRINT TAB(5);OLD.FLD$(0);TAB(15);OLD.FLD$(7)
  PRINT TAB(25);OLD.FLD$(1);" ";OLD.FLD$(2)
  PRINT TAB(25);OLD.FLD$(3)
  PRINT TAB(25);OLD.FLD$(4);" ";OLD.FLD$(5);" ";OLD.FLD$(6)
  PRINT
  CONSOLE
  RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          PAUSE ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF PAUSE%
  PRINT
  INPUT "Press 'RETURN' to continue ---";LINE PAUSE$
  RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM

```

Listing 2-2. (continued)

```

REM      CONVERT TARGET VALUE TO KEY FORMAT ROUTINE
REM
REM :.....:
DEF KEY.FORMAT$(KEY%,TARGET%)
  IF UNIQ.KEY% = KEY% THEN \
    KEY.FORMAT$ = TARGET$ :\
  RETURN
  KL% = KEY.LEN%(KEY%)
  KEY.FORMAT$ = LEFT$(TARGET$ + SPACES$,KL%-2) + \
    CHR$(0) + CHR$(0)
RETURN
FEND

REM :.....:
REM
REM      COMPARE INDEX.KEY & U.VALUE ROUTINE
REM
REM :.....:
DEF COMPARE%
  IF KEY% = UNIQ.KEY% THEN \
    KL% = KEY.LEN%(KEY%) \
  ELSE \
    KL% = KEY.LEN%(KEY%)-2
  C1$ = LEFT$(INDEX.KEY$ + SPACE$,KL%)
  C2$ = LEFT$(U.VALUE$ + SPACE$,KL%)
  IF C1$<C2$ THEN \
    COMPARE% = -1 :\
  RETURN
  IF C1$>C2$ THEN \
    COMPARE% = 1 \
  ELSE \
    COMPARE% = 0
RETURN
FEND

REM :.....:
REM
REM      CHECK LOCK ROUTINES
REM
REM :.....:
DEF SKIP.LOCK%
  WHILE DRN% <> 0 AND LOKCOD <> 0
    L.VALUE$ = LEFT$(INDEX.KEY$,KEY.LEN%(KEY%))
    INDEX.KEY$ = SET.LENGTH$
    DRN% = APTKEY(KEY.NUM%(KEY%),FILE.NO%,SLOCK%, \
      L.VALUE$,INDEX.KEY$)
  WEND
  RETURN
FEND
DEF CHECK.LOCK%
  PRINT
  INPUT \

```

Listing 2-2. (continued)

```

"Enter a 'W' if you want to wait for locked record(s)>>"; \
LINE DUMMY$
IF UCASE$(DUMMY$) = "W" THEN \
  CHECK.LOCK% = YES% : \
  RETURN
WHILE DRN% <> 0 AND LOKCOD <> 0
  CONV.TARGET$ = LEFT$(INDEX.KEY$,KEY.LEN%(KEY%))
  INDEX.KEY$ = SET.LENGTH$
  IF OLD.ACTION$ = "CONT" THEN \
    DRN% = APTKEY(KEY.NUM%(KEY%),FILE.NO%, \
      SLOCK%, CONV.TARGET$,INDEX.KEY$) \
  ELSE \
    DRN% = BEFKEY(KEY.NUM%(KEY%),FILE.NO%, \
      SLOCK%, CONV.TARGET$,INDEX.KEY$)
WEND
CHECK.LOCK% = NO%
RETURN
FEND
DEF SET.XLOCK$(OP$)
30010 IF SETLOK(FILE.NO%,XLOCK%,DRN%) <> 0 THEN \
  PRINT : \
  PRINT "Customer update on hold due to record lock" : \
  INPUT \
"Enter 'W' if you want to wait or press 'RET' to cancel update>>"; \
LINE DUMMY$: \
DUMMY$ = UCASE$(DUMMY$) \
ELSE \
  DUMMY$ = "ok"
IF DUMMY$ = "W" THEN 30010
IF DUMMY$ = "ok" AND OP$ = "S" THEN \
  SET.XLOCK$ = "SAVE"
IF DUMMY$ = "ok" AND OP$ = "D" THEN \
  SET.XLOCK$ = "DELT"
IF DUMMY$ <> "ok" THEN \
  SET.XLOCK$ = OLD.ACTION$
RETURN
FEND
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          CUST # UNIQUENESS TEST ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
DEF TEST.UNIQUENESS%
TEST$ = NEW.FLD$(KEY.MAP%(UNIQ.KEY%))
TEST% = GETKEY(UNIQ.KEY%,0,NLOCK%,TEST$)
IF LOKCOD <> 0 THEN \
  DUMMY% = LOCK.TYPE%(12)
IF TEST% = 0 THEN \
  TEST.UNIQUENESS% = YES% \
ELSE \
  TEST.UNIQUENESS% = NO% : \
  PRINT : \
  PRINT " *** Already Assigned ***" : \

```

**Listing 2-2. (continued)**



```

                PRINT
    RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          UPDATE DATA FIELD ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF UPDATE.FIELD%(FIELD.NO%)
    FIELD.NO% = FIELD.NO%-1
1050    PRINT
        PRINT "Input new ";FLD.NAME$(FIELD.NO%);
        INPUT ">>";LINE NEW.FLD$(FIELD.NO%)

    IF FIELD.NO% = KEY.MAP%(UNIQ.KEY%) THEN \
        NEW.FLD$(FIELD.NO%) = RIGHT$("0000"+NEW.FLD$(FIELD.NO%), \
            FLD.LEN%(FIELD.NO%)) \
    ELSE \
        NEW.FLD$(FIELD.NO%) = LEFT$(NEW.FLD$(FIELD.NO%), \
            FLD.LEN%(FIELD.NO%))

    IF FIELD.NO% <> 0 OR NEW.FLD$(FIELD.NO%) = \
        OLD.FLD$(FIELD.NO%) THEN RETURN
    UNIQUE% = TEST.UNIQUENESS%
    IF NOT UNIQUE% THEN 1050
    RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          WARNING MESSAGES
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF WARNING.TYPE%(TYPE%,RET.CODE%)
    PRINT
    PRINT "WARNING...Return Code #";RET.CODE%; \
        " occurred while trying to ";
    ON TYPE% GOTO 9930,9940,9950
9930    PRINT "remove old key from ";INDEX.NAME$(KEY%)
        DUMMY% = PAUSE% :RETURN
9940    PRINT "enter key into ";INDEX.NAME$(KEY%)
        DUMMY% = PAUSE% :RETURN
9950    PRINT "delete key from ";INDEX.NAME$(KEY%)
        DUMMY% = PAUSE% :RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          ADD NEW KEY VALUE ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

```

Listing 2-2. (continued)

```

DEF ADD.KEY%(KEY%,DRN%)
  K.FLD% = KEY.MAP%(KEY%)
  OLD.KEY$ = OLD.FLD$(K.FLD%)
  NEW.KEY$ = NEW.FLD$(K.FLD%)
REM
REM REMOVE OLD KEY VALUE
REM
  RET.CODE% = DELKEY(KEY.NUM%(KEY%),FILE.NO%, \
    XLOCK%,OLD.KEY$,DRN%)
  IF ERRCOD <> 0 THEN \
    DUMMY% = ERROR.TYPE%(4)
  IF LOKCOD <> 0 THEN \
    DUMMY% = LOCK.TYPE%(6)
  IF RET.CODE% <> 1 THEN \
    DUMMY% = WARNING.TYPE%(1,RET.CODE%)
REM
REM ADD NEW KEY VALUE
REM
  RET.CODE% = ADDKEY(KEY.NUM%(KEY%),FILE.NO%, \
    XLOCK%,NEW.KEY$,DRN%)
  IF ERRCOD <> 0 THEN \
    DUMMY% = ERROR.TYPE%(5)
  IF LOKCOD <> 0 THEN \
    DUMMY% = LOCK.TYPE%(7)
  IF RET.CODE% <> 1 THEN \
    DUMMY% = WARNING.TYPE%(2,RET.CODE%)
  RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          WRITE NEW DATA RECORD ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF WRITE.CUST%(DRN%)
  OUTBUF$ = CHR$(0)          REM CLEAR DELETE FLAG
  FOR D.FLD% = 0 TO MAX.FIELD%
    OUTBUF$ = OUTBUF$ + LEFT$(NEW.FLD$(D.FLD%) + \
      FLD.SPC$, FLD.LEN%(D.FLD%))
  NEXT D.FLD%
  OUTBUF.PTR% = SADD(OUTBUF$) + 2
  IF WRTDAT(FILE.NO%,DRN%,OUTBUF.PTR%) <> 0 THEN \
    DUMMY% = ERROR.TYPE%(12)
  RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          DELETE KEY VALUE FROM INDEX ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF DEL.KEY%(KEY%,DRN%)
  K.FLD% = KEY.MAP%(KEY%)

```

Listing 2-2. (continued)

```

OLD.KEY$ = OLD.FLD$(K.FLD%)
RET.CODE% = DELKEY(KEY.NUM%(KEY%),FILE.NO%, \
XLOCK%,OLD.KEY$,DRN%)
IF ERRCOD <> 0 THEN \
    DUMMY% = ERROR.TYPE%(6)
IF LOKCOD <> 0 THEN \
    DUMMY% = LOCK.TYPE%(10)
IF RET.CODE% <> 1 THEN \
    DUMMY% = WARNING.TYPE%(3,RET.CODE%)
RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          DATA ENTRY ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF ENTER.DATA$(ENTER.MODE$)
IF ENTER.MODE$ = "NEW" THEN \
    FOR FLD% = 0 TO MAX.FIELD% : \
        OLD.FLD$(FLD%) = "" : \
    NEXT FLD%
IF ENTER.MODE$ = "OLD" THEN \
    FOR FLD% = 0 TO MAX.FIELD% : \
        NEW.FLD$(FLD%) = OLD.FLD$(FLD%) : \
    NEXT FLD%
DUMMY% = CLEAR.SCREEN%
WHILE ENTER.MODE$ = "NEW"
PRINT TAB(20);"Enter New Customer Information"
PRINT TAB(20);" *****"
PRINT :PRINT
PRINT TAB(5); \
    "[Press 'RETURN' for customer # to see main menu.]"
PRINT
FOR FLD% = 0 TO MAX.FIELD%
    FLD.NO% = FLD% + 1
1010 PRINT TAB(4);FLD.NO%;"- ";FLD.NAME$(FLD%); \
    TAB(30);"(";FLD.LEN$(FLD%);")";TAB(38);
    INPUT ">>";LINE NEW.FLD$(FLD%)
    IF FLD% = 0 AND NEW.FLD$(FLD%) = "" THEN \
        ENTER.DATA$ = "STOP" : \
        RETURN

    IF FLD% = KEY.MAP%(UNIQ.KEY%) THEN \
        NEW.FLD$(FLD%) = RIGHT$("0000"+NEW.FLD$(FLD%), \
            FLD.LEN$(FLD%)) : \
        UNIQUE% = TEST.UNIQUENESS% \

ELSE \
    NEW.FLD$(FLD%) = LEFT$(NEW.FLD$(FLD%), \
        FLD.LEN$(FLD%)) : \
    UNIQUE% = YES%

    IF NOT UNIQUE% THEN GOTO 1010
NEXT FLD%

```

Listing 2-2. (continued)

```

        ENTER.MODE$ = "NEWMOD"
WEND
1015 PRINT :PRINT :PRINT
    PRINT TAB(20);"Current customer information" : PRINT
    FOR FLD% = 0 TO MAX.FIELD%
        FLD.NO% = FLD% + 1
        PRINT TAB(4);FLD.NO%;"- ";FLD.NAME$(FLD%);TAB(30); \
            NEW.FLD$(FLD%)
    NEXT FLD%
REM NEW DATA HAS FEWER OPTIONS
    IF ENTER.MODE$ = "NEWMOD" THEN 1030
1020 PRINT :PRINT
    PRINT \
"Press 'RETURN' to continue scan, enter Field # to change data,"
    PRINT \
"S to save changes, D to delete data, B for back scan, or E"; \
    " " to end scan";
    INPUT ">>";LINE OP$
    OP$ = UCASE$(OP$)
    IF OP$ = "" THEN ENTER.DATA$ = "CONT":RETURN
    IF OP$ = "S" THEN ENTER.DATA$ = SET.XLOCK$(OP$):RETURN
    IF OP$ = "D" THEN ENTER.DATA$ = SET.XLOCK$(OP$):RETURN
    IF OP$ = "B" THEN ENTER.DATA$ = "BACK":RETURN
    IF OP$ = "E" THEN ENTER.DATA$ = "STOP":RETURN
    OP% = VAL(OP$)
    IF OP% < 1 OR OP% > NO.FIELDS% THEN 1020
    DUMMY% = UPDATE.FIELD$(OP%)
    GOTO 1015 REM DISPLAY INFO
1030 PRINT :PRINT
    PRINT \
"Press 'RETURN' to save data, enter Field # to change data,"
    INPUT "D to delete data, or E to end input>>";LINE OP$
    OP$ = UCASE$(OP$)
    IF OP$ = "" OR OP$ = "S" THEN ENTER.DATA$ = "SAVE":RETURN
    IF OP$ = "D" THEN ENTER.DATA$ = "DELT":RETURN
    IF OP$ = "E" THEN ENTER.DATA$ = "STOP":RETURN
    OP% = VAL(OP$)
    IF OP% < 1 OR OP% > NO.FIELDS% THEN 1030
    DUMMY% = UPDATE.FIELD$(OP%)
    GOTO 1015
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM UPDATE INDICES & DATA FILE ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF UPDATE$(DATA.RECORD%)
    IF DATA.RECORD% = 0 THEN \
        DATA.RECORD% = NEWREC(FILE.NO%,XLOCK%)
    UPDATE% = DATA.RECORD%
    IF ERRCOD <> 0 THEN \
        DUMMY% = ERROR.TYPE%(8)
    IF LOKCOD <> 0 THEN \

```

Listing 2-2. (continued)

```

        DUMMY% = LOCK.TYPE%(3)
    FOR KEY% = 0 TO MAX.KEY%
        FLD% = KEY.MAP%(KEY%)
        IF OLD.FLD$(FLD%) <> NEW.FLD$(FLD%) THEN \
            DUMMY% = ADD.KEY%(KEY%,DATA.RECORD%)
    NEXT KEY%
    FOR FLD% = 0 TO MAX.FIELD%
        IF OLD.FLD$(FLD%) <> NEW.FLD$(FLD%) THEN \
            DUMMY% = WRITE.CUST%(DATA.RECORD%) :\
            RETURN
    NEXT FLD%
    RETURN
FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          DELETE INDEX & DATA FILE ENTRY ROUTINE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

DEF DELETE%(DATA.RECORD%)
    FOR KEY% = 0 TO MAX.KEY%
        FLD% = KEY.MAP%(KEY%)
        IF OLD.FLD$(FLD%) <> "" THEN \
            DUMMY% = DEL.KEY%(KEY%,DATA.RECORD%)
    NEXT KEY%
    IF RETREC(FILE.NO%,XLOCK%,DATA.RECORD%) <> 0 THEN \
        DUMMY% = ERROR.TYPE%(9)
    IF LOKCOD <> 0 THEN \
        DUMMY% = LOCK.TYPE%(9)
    RETURN
FEND

REM          END OF UTILITY FUNCTIONS
REM
REM ++++++
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          INITIALIZE INDEX FILES
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
2000
    SET.LENGTH$ = "12345678901"
    INDEX.KEY$ = SET.LENGTH$
    SPACE$ = " "

REM
REM SET TERMINAL TO -1 FOR AUTOMATIC ASSIGNMENT BY AM80
REM
    TERMINAL% = -1
    TRAP.ERRORS% = YES%
    TIME.OUT.TEST.DELAY% = 2 REM APPROXIMATELY 2 SECONDS
    TERMINAL% = INTUSR(TERMINAL%,TRAP.ERRORS%,TIME.OUT.TEST.DELAY%)
    IF ERRCOD <> 0 THEN \
        DUMMY% = ERROR.TYPE%(14)

```

**Listing 2-2. (continued)**

```

NO.BUFFERS% = 5
NO.NODE.SECTORS% = 4
NO.DATA.FILES% = 1
NO.KEYS% = MAX.KEY% + 1
IF SETUP(NO.BUFFERS%,NO.KEYS%,NO.NODE.SECTORS%, \
NO.DATA.FILES%) <> 0 THEN \
    PRINT "Illegal SETUP Parameters" : \
    STOP

FOR KEY% = 0 TO MAX.KEY%
    KEY.NUM%(KEY%) = OPNIDX(-1,INDEX.NAME$(KEY%), \
    KEY.LEN%(KEY%), KEY.TYPE%(KEY%),KEY.DUP%(KEY%))
    IF ERRCOD <> 0 THEN \
        DUMMY% = ERROR.TYPE%(1)
NEXT KEY%

REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM      '      INITIALIZE DATA FILE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM      FILE.NO% = -1
REM      RECORD.LENGTH% = 100
REM      FILE.NAME$ = "CUSTOMER.DAT"
REM      FILE.NO% = OPNDAT(FILE.NO%,SFILE%,FILE.NAME$,RECORD.LENGTH%)
REM      IF ERRCOD <> 0 THEN \
REM          DUMMY% = ERROR.TYPE%(10)
REM      IF LOKCOD <> 0 THEN \
REM          DUMMY% = LOCK.TYPE%(1)
4990  REM INITIALIZE STRING UTILITIES
TMPBUF$ = "12345678901234567890123456789012345678901234567890"

INPBUF$ = TMPBUF$ + TMPBUF$

REM
REM INPBUF IS THE BUFFER AREA FOR THE READAT ROUTINE
REM
REM      INPBUF.PTR% = SADD(INPBUF$) + 2

REM      123456789012345678901234567890123456
REM      FLD.SPC$ = "      "

REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM      BEGIN DATABASE OPERATION
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
5000  DUMMY% = CLEAR.SCREEN%
      CHOICE% = MAIN.MENU%
      ON CHOICE% GOTO 5100,5300,5500,5700,5900,6100

REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM      ENTER NEW CUSTOMERS

```

Listing 2-2. (continued)

```

REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
5100 ACTION$ = ENTER.DATA$("NEW")
      LOCK.CODE% = 0
      IF ACTION$ = "SAVE" THEN \
          NDRN% = UPDATE%(0): \ UPDATE INDICES & DATA FILE
          LOCK.CODE% = FRELOK(FILE.NO%,XLOCK%,NDRN%)
      IF LOCK.CODE% <> 0 THEN \
          DUMMY% = LOCK.TYPE%(8)
      IF ACTION$ = "SAVE" THEN \
          GOTO 5100 \
      ELSE \
          GOTO 5000 REM RETURN TO MENU

REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM SCAN/UPDATE/DELETE CUSTOMERS
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
5300 KEY% = SEARCH.KEY% REM DETERMINE SEARCH KEY
      PRINT
      PRINT "Enter target value for ";KEY.NAME$(KEY%);","
      INPUT " or press 'RETURN' to see main menu>>"; \
          LINE TARGET$
      IF TARGET$ = "" THEN 5000
      CONV.TARGET$ = KEY.FORMAT$(KEY%,TARGET$)
5345 DRN% = SERKEY(KEY.NUM%(KEY%),FILE.NO%,SLOCK%, \
          CONV.TARGET$,INDEX.KEY$)
      IF ERRCOD <> 0 THEN \
          DUMMY% = ERROR.TYPE%(2)
      IF LOKCOD <> 0 THEN \
          STAYPUT% = CHECK.LOCK% \
      ELSE \
          STAYPUT% = NO%
      IF STAYPUT% THEN 5345
      OLD.ACTION$ = "CONT"
      CONTINUE% = YES%
      WHILE CONTINUE% AND DRN% <> 0
          LDRN% = DRN% REM save drn for lock release
          DUMMY% = READ.CUST%(DRN%)
          ACTION$ = ENTER.DATA$("OLD")
          SAVE.KEY% = KEY%
          IF ACTION$ = "SAVE" THEN \
              DUMMY% = UPDATE%(DRN%)
          IF ACTION$ = "DELT" \
              THEN DUMMY% = DELETE%(DRN%)
          IF ACTION$ <> "DELT" AND FRELOK(FILE.NO%,RLOCK%,LDRN%) <> 0 \
              THEN DUMMY% = LOCK.TYPE%(2)
          IF ACTION$ = "SAVE" OR ACTION$ = "DELT" THEN \
              KEY% = SAVE.KEY% : \ RESET SEARCH KEY
              ACTION$ = OLD.ACTION$ REM reset direction
          OLD.ACTION$ = ACTION$
          CONV.TARGET$ = LEFT$(INDEX.KEY$,KEY.LEN%(KEY%))
          INDEX.KEY$ = SET.LENGTH$
          LOCK.CODE% = 0

```

**Listing 2-2. (continued)**

```

5390      IF ACTION$ = "CONT" THEN \
          DRN% = APTKEY(KEY.NUM%(KEY%),FILE.NO%, \
                    SLOCK%, CONV.TARGET$,INDEX.KEY$) :\
          LOCK.CODE% = LOKCOD
IF ACTION$ = "BACK" THEN \
          DRN% = BEFKEY(KEY.NUM%(KEY%),FILE.NO%, \
                    SLOCK%, CONV.TARGET$,INDEX.KEY$):\
          LOCK.CODE% = LOKCOD
IF LOCK.CODE% <> 0 THEN \
          STAYPUT% = CHECK.LOCK% \
ELSE \
          STAYPUT% = NO%
IF STAYPUT% THEN 5390
IF ACTION$ = "STOP" THEN \
          CONTINUE% = NO%

WEND
PRINT
PRINT "SCAN ENDED"
DUMMY% = PAUSE%
GOTO 5000 REM RETURN TO MAIN MENU

REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          LIST CUSTOMERS
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
5500      KEY% = SEARCH.KEY%
          PRINT
          INPUT \
"do you want listing routed to printer (Y/N)>>";LINE ROUTE$
          ROUTE$ = UCASE$(ROUTE$)
          PRINT
          PRINT \
"Enter lower and upper limits for ";KEY.NAME$(KEY%);" listing;"
          INPUT \
"
          separate values with a comma >>";L.VALUE$,U.VALUE$
          L.VALUE$ = KEY.FORMAT$(KEY%,L.VALUE$)
          U.VALUE$ = KEY.FORMAT$(KEY%,U.VALUE$)
          DRN% = SERKEY(KEY.NUM%(KEY%),FILE.NO%,SLOCK%, \
                    L.VALUE$,INDEX.KEY$)
          IF LOKCOD <> 0 THEN \
              DUMMY% = SKIP.LOCK%
          NO.LISTED% = 0
          WHILE DRN% <> 0 AND COMPARE%<= 0
              DUMMY% = READ.CUST%(DRN%)
              DUMMY% = PRINT.CUST%
              NO.LISTED% = NO.LISTED% + 1
              IF FRELOK(FILE.NO%,SLOCK%,DRN%) <> 0 THEN \
                  DUMMY% = LOCK.TYPE%(4)
              L.VALUE$ = LEFT$(INDEX.KEY$,KEY.LEN%(KEY%))
              INDEX.KEY$ = SET.LENGTH$
              DRN% = APTKEY(KEY.NUM%(KEY%),FILE.NO%,SLOCK%, \
                    L.VALUE$,INDEX.KEY$)
              IF LOKCOD <> 0 THEN \
                  DUMMY% = SKIP.LOCK%

```

Listing 2-2. (continued)



```

WEND
IF DRN% <> 0 THEN \
  LOCK.CODE% = FRELOK(FILE.NO%,SLOCK%,DRN%) \
ELSE \
  LOCK.CODE% = 0
IF LOCK.CODE% <> 0 THEN \
  DUMMY% = LOCK.TYPE%(5)
PRINT
PRINT TAB(5);NO.LISTED%;" records listed."
DUMMY% = PAUSE%
GOTO 5000 REM RETURN TO MAIN MENU

REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          DATABASE STATISTICS
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
5700  DUMMY% = CLEAR.SCREEN%
      PRINT TAB(5);FILE.NAME$;" has ";GETDFS(FILE.NO%); \
        " records; currently, ";
      PRINT GETDFU(FILE.NO%)," of them are in use."
      PRINT :PRINT :PRINT :PRINT
      PRINT TAB(5);"INDEX";TAB(30);"ENTRIES"
      PRINT TAB(5);"-----";TAB(30);"-----"
      FOR KEY% = 0 TO MAX.KEY%
        PRINT TAB(5);KEY.NAME$(KEY%);TAB(32);NOKEYS(KEY%)
      NEXT KEY%
      PRINT :PRINT :PRINT :PRINT
      DUMMY% = PAUSE%
      GOTO 5000 REM RETURN TO MAIN MENU

REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          SAVE DATABASE UPDATES & RESTART
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
5900  IF SAVDAT(FILE.NO%) <> 0 THEN \
      DUMMY% = ERROR.TYPE%(7)
      FOR KEY% = 0 TO MAX.KEY%
        IF SAVIDX(KEY.NUM%(KEY%)) <> 0 THEN \
          DUMMY% = ERROR.TYPE%(3)
      NEXT KEY%
      GOTO 5000

REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM          SAVE DATABASE UPDATES & TERMINATE
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
6100  IF CLSDAT(FILE.NO%) <> 0 THEN \
      DUMMY = ERROR.TYPE%(15)
      FOR KEY% = 0 TO MAX.KEY%
        IF CLSIDX(KEY.NUM%(KEY%)) <> 0 THEN \
          DUMMY% = ERROR.TYPE%(16)
      NEXT KEY%
      IF FRELOK(FILE.NO%,SFILE%,0) <> 0 THEN \
        DUMMY% = ERROR.TYPE%(13)
      PRINT
      PRINT " *** SUCCESSFUL TERMINATION ***"
      STOP

```

Listing 2-2. (continued)

## 2.7 CB80 Source Code Data File Routines

The CB80 data file routines presented in this section can be used to

- Open a data file.
- Close a data file.
- Get space for new records.
- Report the size of a data file.
- Return space from deleted records.

**Note:** use CB80 routines only in single-user environments because the Access Manager data locking facilities apply only to data file routines. Furthermore, the routines to keep track of deleted records and determine the next available record are not designed for multiuser environments.

### 2.7.1 OPEN.DATA.FILE%(DATA.FILE\$,FILE.NO%,RECORD.LENGTH%)

This routine opens DATA.FILE\$ with the specified RECORD.LENGTH% as FILE.NO%. If DATA.FILE\$ does not exist, it is created. The RECORD.LENGTH% must be the same each time the file is opened and it must be at least nine bytes.

**Note:** no data can be stored in the first record of DATA.FILE\$. The first record is a header that maintains the status of the DATA.FILE\$ stack structure.

FILE.NO% is constrained by the limitations of CB80 because DATA.FILE\$ is a standard CB80 disk file.

```
DUMMY%=OPEN.DATA.FILE%("D:CUSTOMER.DAT",4,128)
```

The above call opens the specified data file as file number four with a record length of 128 bytes. If CUSTOMER.DAT does not exist on drive D, it is created.

### 2.7.2 CLOSE.DATA.FILE%(FILE.NO%)

This routine closes data files previously opened with a call to OPEN.DATA.FILE%. It must be used to close such files or the header record will be improperly updated.

```
DUMMY%=CLOSE.DATA.FILE%(4)
```

### 2.7.3 NEW.DATA%(FILE.NO%)

This routine returns the record number of the next available record in the DATA.FILE\$ opened as FILE.NO%. The next available record is removed from the top of the stack for FILE.NO%. If the stack is empty, NEW.DATA% automatically increments the size of the data file to generate space for a new record. For example,

```

DUMMY%=OPEN.DATA.FILE%("D:CUSTOMER.DAT",4,128)
INPUT "Enter new customer number and last name:"; \
    CUST.NO, LAST.NAME$
DRN%=NEW.DATA%(4)
CUST.NO$=CONVERT$(CUST.NO,2)
RET.CODE%=ADDKEY(0,0,0,CUST.NO$,DRN%)
IF ERRCOD<>0 THEN DUMMY%=TROUBLE%(4)
IF RET.CODE%<>1 THEN DUMMY%=WARNING%(4,RET.CODE%)
RET.CODE%=ADDKEY(1,0,0, LAST.NAME$,DRN%)
IF ERRCOD<>0 THEN DUMMY%=TROUBLE%(5)
IF RET.CODE%<>1 THEN DUMMY%=WARNING%(5,RET.CODE%)
PRINT #4,DRN%;CUST.NO, LAST.NAME$

```

In the preceding example, DRN% is set to the value of an empty record in the customer data file. A new customer number and last name are then added to the corresponding B-Trees with the associated record number given by DRN%. Finally, record DRN% in the customer data file is initialized.

If a user error occurs, control transfers to TROUBLE% for appropriate action. The parameter in TROUBLE% is a code to let the function know where the user error occurred. Control passes to WARNING% if the ADDKEY functions are not successful. Both the location code and the RET.CODE% pass to WARNING%. Note that you must provide both of these routines because they are not part of Access Manager.

#### 2.7.4 RETURN.DATA%(FILE.NO%,DATA.RECORD%,MESSAGE\$)

This routine pushes the returned DATA.RECORD% to the top of the stack for FILE.NO%. Deleted data records are organized according to a stack structure, that is, last-in/first-out (LIFO). After a call to RETURN.DATA%, the record in the data file with record number DATA.RECORD% has these two fields overwritten:

Table 2-1. Data Record Fields

Field	Data
1	Link to next available data record.
2	MESSAGE\$.

You can use the MESSAGE\$ parameter to flag deleted (returned) records or to save information from the deleted record for subsequent processing.

You might find it necessary to add enough trailing commas to MESSAGE\$ so a deleted record has a full complement of data fields. This allows returned records to be read with the same read statements as regular records. This is shown in the following code segment:

```

INPUT "Enter customer number: ";CUST.NO
CUST.NO$=CONVERT$(CUST.NO,2)
DRN%=GETKEY(0,0,0,CUST.NO$)
IF DRN%<>0 THEN \
  IF DELKEY(0,0,0,CUST.NO$,DRN%) = 1 THEN \
    DUMMY%=RETURN.DATA%(4,DRN%,"DELETED,,,,,,")

```

In the preceding example, the data record corresponding to CUST.NO\$ returns to the data file for future use and the customer number is deleted from the index file, unless no such customer number exists.

#### 2.7.5 DATA.FILE.SIZE%(FILE.NO%)

This routine returns the total number of records used by a data file, including the header record and any returned but unused records.

#### 2.7.6 DATA.FILE.UTILIZATION%(FILE.NO%)

This routine returns a count of the number of records in a data file currently being used to store data. It excludes the header record and any returned but unused records.

End of Section 2

## Section 3

# Using Access Manager with PL/I-80 Applications

This section contains instructions for implementing Access Manager with application programs coded in PL/I-80.

Two examples are provided in this section. The first shows how to use many of the Access Manager functions described in Section 3 of your Reference Manual, and how to use the data file routines in your PL/I-80 applications. The second example provides an extensive illustration of using Access Manager to construct and maintain a data base.

### 3.1 Linking Access Manager to Your Application Program

This section discusses a PL/I-80 application program called MYPROG that you write and compile to produce a binary relocatable file.

#### 3.1.1 Linking Single-user PL/I-80 Applications

You must link your compiled application program to the appropriate Access Manager subroutine library and index file buffer module. The following command line can be used to create an executable version of MYPROG:

```
LINK MYPROG,AM80PLI.IRL[S,A],AM80BUF.IRL
```

AM80BUF contains the buffer area beginning with entry point AM8FCB and ending with AM8END.

Before linking, be sure AM80BUF is large enough to contain your buffers (as specified in the SETUP function). You can use SETAMBUF to create a correctly sized buffer module.

#### 3.1.2 Linking Multiuser PL/I-80 Applications

If your single-user version of MYPROG is coded with appropriate data locking procedures, you do not have to recompile it to create a multiuser version. All that is necessary is to relink the program.

You must link your compiled application program to the appropriate Access Manager multiuser interface. The interface makes the queue calls to the shared code in the background server. The background server resides in its own memory segment.

To create a PRL file that calls the Access Manager background server, use LINK.COM as follows:

```
LINK MYPROG[OP,A],AMQ8PLI.IRL
```

### 3.2 External Declaration of Access Manager Routines

PL/I-80 requires that external routines (those not coded in the program module but referenced by it) be explicitly declared. The file AM80EXTR.PLI contains external function declarations for the entire set of Access Manager routines. Use the %INCLUDE feature of PL/I-80 to make these external declarations a part of your application program.

Note that AM80EXTR.PLI expects two compile-time constants to be defined with the %REPLACE macro of PL/I-80. NAME\_LEN (the maximum length of index and data filenames) and MAX\_KEY\_LEN (the maximum key value length) must be set to appropriate values before AM80EXTR.PLI is included. For example, the following code segment can be used:

```
%REPLACE
    NAME_LEN BY 14,
    MAX_KEY_LEN BY 48;

%INCLUDE 'AM80EXTR.PLI';
```

It is not necessary to set MAX\_KEY\_LEN at the Access Manager maximum of 48. Any value less than or equal to 48 that is sufficient for your particular application is valid.

**Note:** consider these points concerning the passing of parameters between Access Manager and PL/I-80:

- Access Manager requires all string-valued parameters (FILENAME, IDXNAME, KEYVAL, and IDXVAL) to be declared as CHARACTER() VARYING. CHARACTER VARYING strings in PL/I-80 reserve the leading byte for a length counter Access Manager uses to determine the actual length of a string-valued parameter.
- The output string parameter IDXVAL must be passed by reference, as opposed to value. Therefore, the actual parameter passed to Access Manager (for IDXVAL) must be declared with exactly the same attributes as the formal IDXVAL parameter in AM80EXTR.PLI. This implies the actual variable used for IDXVAL (ACTUAL\_IDXVAL) must be declared as follows:

```
DCL
    ACTUAL_IDXVAL CHAR(MAX_KEY_LEN) VAR;
```

### 3.3 Coding Numeric Key Values

For a general discussion of coding numeric key values, refer to the ADDKEY function description in Section 3 of your Reference Manual.

In the PL/I environment, the easiest approach to represent numeric key values is to use FIXED DECIMAL quantities. Because FIXED DECIMAL quantities are stored in BCD (Binary Coded Decimal) format with the least significant byte first and the sign bit set in the last byte, KEYTYP% should be one. Access Manager requires a CHARACTER VARYING value for the KEYVAL\$ and IDXVAL\$ parameters. Therefore, FIXED DECIMAL variables should be overlaid (based) on KEYVAL\$ and IDXVAL\$ string variables.

The key length is based on the number of bytes required to store the FIXED DECIMAL quantities. A FIXED DECIMAL with 'p' digits requires:

$$\text{INT}((p + 2) / 2)$$

bytes where INT returns the integer portion of its argument. For example, a FIXED DECIMAL quantity with eight digits requires five bytes of storage.

The following declarations and assignments permit the use of FIXED DECIMAL quantities as key values in Access Manager.

```
%REPLACE
  MAX_KEY_LEN BY 48,
  NAME_LEN BY 14,
  P BY 8,           /* example precision */
  Q BY 2,           /* fractional places */
  KEYLEN BY 5;     /* INT((P+2)/2) */

%INCLUDE 'AM80EXTR.PLI';

DCL
  (KEYVAL,IDXVAL) CHAR(MAX_KEY_LEN) VAR,
  (BCDINP_PTR,BCDOUT_PTR) PÖINTER;

DCL
  1 BCDINP BASED (BCDINP_PTR),
    2 LEN FIXED BINARY (7),
    2 VAL FIXED DECIMAL (P,Q),

  1 BCDOUT BASED (BCDOUT_PTR),
    2 LEN FIXED BINARY (7),
    2 VAL FIXED DECIMAL (P,Q);
```

```
BCDINP_PTR = ADDR(KEYVAL); /* overlay bcd on string */
BCDINP.LEN = KEYLEN;      /* set length byte
                           of string */

BCDOUT_PTR = ADDR(IDXVAL);
BCDOUT.LEN = KEYLEN;
```

Whenever you use a numeric quantity with an Access Manager function, use KEYVAL for input values and IDXVAL for output values. To manipulate the key as a numeric quantity, refer to BCDINP.VAL and BCDOUT.VAL for input and output key values, respectively. For example,

```
BCDINP.VAL = 123.45;
DRN = SERKEY(KEY_NO,DFILE,DLOCK,KEYVAL,IDXVAL);
IF DRN ~= 0 | DATVAL() ~= 0 THEN
    PUT SKIP LIST (BCDOUT.VAL);
```

prints the numeric value of the first key value in the index greater than or equal to 123.45, unless no such key exists.

### 3.4 Using the RECREATE.PLI Utility Program

RECREATE.PLI contains the PL/I-80 source code for the RECREATE utility program. You can change the source code in whatever way you want. To create RECREATE.COM, compile RECREATE.PLI using PLI.COM and then link as follows:

```
LINK RECREATE,AM80PLI.IRL[S,A],AM80BUF.IRL
```

The buffer area for RECREATE is 4,600 bytes based on these parameter values:

- NNSEC% = 4
- NBUFS% = 8
- NDATA% = 1
- NKEYS% = 1

Note that only one data file and one index file are open at the same time RECREATE is running. Use SETAMBUF to configure AM80BUF.IRL.

Table 3-1 shows the layout and content of records in a Recreate Parameter File. This particular example file can be used to reconstruct DATABASE (see Listing 3-2).



**Table 3-1. Example PL/I-80 Recreate Parameter File**

Record Type	Contents
Header	1,4
Data File	CUSTOMER.DAT,100,3,0
Index File	NAME.IDX,10,0,1,1,Y
Key Part	22,8
Index File	NUMB.IDX,4,0,0,1,N
Key Part	2,4
Index File	ZIPC.IDX,11,0,1,1,Y
Key Part	84,9

If you want to change the capacities of the RECREATE program (and hence its memory requirements), note the following key constants:

- **MAX\_NO\_KEYS** specifies the maximum number of index files associated with a data file; **MAX\_KEY\_PARTS** indicates the maximum number of fields comprising a key value.
- **MAX\_SORT** is the maximum number of key values that can be buffered by RECREATE.PLI before being sorted and added to the index file being recreated.
- **MAX\_SPACE** specifies the actual number of bytes available for the buffered key values. Each key value requires one more byte than its key length.

The actual number of buffered key values depends on the key length. For short key lengths, **MAX\_SORT** will be the limiting factor. **MAX\_SPACE** is the limiting factor for long key lengths.

The constant **MAX\_REC\_LEN** should be increased if your applications require data files with record lengths exceeding 1024 bytes.

### 3.5 PL/I-80 Data File Example

The following listing illustrates use of the primary Access Manager functions to update records in a data file:

```

EXAMPLE:
      PROC OPTIONS (MAIN);

%REPLACE
      MAX KEY_LEN BY 48,
      NAME_LEN BY 14;

/* -----
      AM80 External Declarations
----- */
%INCLUDE 'AM80EXTR.PLI';

/* -----
      Exception Processing Routines
----- */
ERROR_HANDLER:
      PROC (LOCALE);
DCL
      LOCALE FIXED;
      PUT SKIP EDIT ('ERROR at ',LOCALE,' with code ',ERRCOD())
          (A,F(3),A,F(4));
      STOP;
END ERROR_HANDLER;

LOCK_CONFLICT:
      PROC (LOCALE)
DCL
      LOCALE FIXED;
      PUT SKIP EDIT ('LOCK Conflict at ',LOCALE,' with code ',ERRCOD())
          (A,F(3),A,F(4));
      STOP;
END LOCK_CONFLICT;

```

**Listing 3-1. PL/I-80 Data File Example**

```

/* -----
   Variable Declarations
----- */
DCL
(N_LOCK,S_LOCK,X_LOCK,S_FILE,X_FILE) FIXED,
(NBUF,NKEYS,NNSEC,NDATF,ERROPT,PROGID,TIMOUT) FIXED,
(DRN,DRN2,FILE_NO,RECORD_LEN) FIXED,
FILE_NAME CHAR(NAME_LEN) VAR;

DCL
1 DAT_BUFFER
  2 PART_NO CHAR(4)
  2 PART_NAME CHAR(20)
  2 PART_QUAN FIXED DECIMAL (15,2),
DATBUF_PTR POINTER;

/* -----
   Lock Parameter Setup
----- */
N_LOCK = 0; /* No lock request */
S_LOCK = 1; /* Shared record lock */
X_LOCK = 2; /* Exclusive record lock */
S_FILE = 3; /* Shared file lock */
X_FILE = 4; /* Exclusive file lock */

/* -----
   System Initialization Parameters
----- */
NBUF = 3; /* 3 buffers */
NKEYS = 1; /* 1 index file */
NNSEC = 4; /* 512-byte index file record length */
NDATF = 1; /* 1 data file */
ERROPT = 1; /* Trap user errors */
PROGID = -1; /* Program ID assigned to MP/M console no. */
TIMOUT = 3; /* Background server time-out delay */

/* -----
   Initialize System
----- */
PROGID = INTUSR(PROGID,ERROPT,TIMOUT);
IF ERRCOD () ~= 0 THEN
  CALL ERROR_HANDLER(1);
IF SETUP(NBUF,NKEYS,NNSEC,NDATF) ~= 0 THEN
  CALL ERROR_HANDLER(2);

```

Listing 3-1. (continued)

Access Manager Programmer's Guide 3.5 PL/I-80 Data File Example

```

/* -----
   Open Files
----- */
FILE NO = -1 /* Automatic file number assignment */
RECORD_LEN = 32;
FILE_NAME = 'K:PART.DAT';
FILE NO = OPNDAT(FILE NO,S_FILE,FILE_NAME,RECORD_LEN);
IF ERRCOD () ~= 0 THEN
    CALL ERROR_HANDLER(3);
IF LOKCOD () ~= 0 THEN
    CALL LOCK_CONFLICT(3);

/* -----
   Initialize Data Buffer Pointer
----- */
DATBUF_PTR = ADDR(DAT_BUFFER);

/* -----
   Set Exclusive Lock on Data Record No. 65686
----- */
DRN2 = 1;
CALL SETDAT(DRN2); /* Set two high-order bytes to 1,
                   which implies a base of 65536 */
DRN = 150; /* 65686 = 65536 + 150 */
IF SETLOK(FILE NO,DRN,DATBUF_PTR) ~= 0 THEN
    CALL LOCK_CONFLICT(4);

/* -----
   Read Data Record
----- */
CALL SETDAT(DRN2);
IF READAT(FILE NO,DRN,DATBUF_PTR) ~= 0 THEN
    CALL ERROR_HANDLER(4);

/* -----
   Update Data Record
----- */
PART_QUAN = PART_QUAN - 100.00;

/* -----
   Write Updated Data Record
----- */
CALL SETDAT(DRN2);
IF WRDAT(FILE NO,DRN,DATBUF_PTR) ~= 0 THEN
    CALL ERROR_HANDLER(5);

```

Listing 3-1. (continued)

```

/* -----
   Release Record Lock
-----
*/
CALL SETDAT(DRN2);
IF FRELOK(FILE_NO,X_LOCK,DRN) ^= 0 THEN
  CALL LOCK_CONFLICT(6);

/* -----
   Close Data File and Release File Lock
-----
*/
IF CLSDAT(FILE_NO) ^= 0 THEN
  CALL ERROR_HANDLER(7);
IF FRELOK(FILE_NO,S_FILE,0) ^= 0 THEN
  CALL LOCK_CONFLICT(7);

END EXAMPLE;

```

### Listing 3-1. (continued)

## 3.6 PL/I-80 DATABASE Source Code

Your Access Manager distribution disk contains sample code for building and maintaining a data base in PL/I-80. The code is designed so you can add or substitute your own key attributes as required. The sample code is on your distribution disk in a file called DATABASE. Note that DATABASE is comprised of three separate components:

- DATABAS1.PLI,
- DATABAS2.PLI,
- DATABASE.DCL.

DATABASE demonstrates the integration of Access Manager with PL/I-80 applications. It builds a name and address data base and provides facilities for examining, updating, and/or listing the information contained therein. You might also want to use routines from DATABASE directly in your application programs.

[SINGLE] To create DATABASE.COM, compile DATABAS1.PLI and DATABAS2.PLI with PLI.COM and link as follows:

```
LINK DATABASE=DATABAS1,DATABAS2,
      AM80PLI.IRL[S,A],AM80BUF.IRL
```

[MULTI] In a multiuser environment, enter the link statement as follows:

```
LINK DATABASE=DATABAS1,DATABAS2,AMQ8PLI.IRL[OP,A]
```

Note that Listing 3-2 of DATABASE might not include recent changes. You should always treat the copy on your distribution disk as the definitive version.

```

                                DATABASE.DCL

/* .....
   DATABASE EXAMPLE DECLARATIONS VERSION 1.05 4/20/82 0913
   .....
*/

%REPLACE
    MAX_KEY BY 2,
    MAX_FIELD BY 7,
    MAX_KEY_LEN BY 20,
    MAX_FLD_LEN BY 20,
    NAME_LEN BY 14,
    FLD_NAME_LEN BY 18,
    ACTION_LEN BY 4,
    NEW_MODE BY 1,
    OLD_MODE BY 2,
    YES BY 1,
    YESBIT BY '1'B,
    NOBIT BY '0'B,
    NO BY 0;

/*
   WORKING VARIABLES
*/

DCL
    (KEY,TERMINAL,TRAP_ERRORS,TIME_OUT_TEST_DELAY,NO_BUFFERS,
     NO_NODE_SECTORS,NO_DATA_FILES,NO_KEYS,FILE_NO,
     RECORD_LENGTH) FIXED STATIC EXTERNAL,
    (SET_LENGTH,IDX_KEY,SPACE) CHAR(MAX_KEY_LEN) VAR STATIC EXTERNAL,
    (SYSLST,SYSCON) FILE,
    OLD_ACTION CHAR(ACTION_LEN) STATIC EXTERNAL,
    FILNAME CHAR(NAME_LEN) VAR STATIC EXTERNAL;

/*
   .....
   DATABASE FIELD & KEY DESCRIPTORS
   .....
*/

DCL
    FLD_NAME(0:MAX_FIELD) CHAR(FLD_NAME_LEN) VAR STATIC EXTERNAL,
    FLD_LEN(0:MAX_FIELD) FIXED BINARY(7) STATIC EXTERNAL,
    (OLD_FLD,NEW_FLD) (0:MAX_FIELD) CHAR(MAX_FLD_LEN) VAR STATIC EXTERNAL,
    NO_FIELDS FIXED STATIC EXTERNAL;

DCL
    IDX_NAME(0:MAX_KEY) CHAR(NAME_LEN) VAR STATIC EXTERNAL,
    KEY_NAME(0:MAX_KEY) CHAR(FLD_NAME_LEN) VAR STATIC EXTERNAL,

```

Listing 3-2. DATABASE.BAS Source Code

```
(KEY_LEN,KEY_MAP,KEY_TYPE,KEY_NUM,KEY_DUP) (0:MAX_KEY) FIXED
      STATIC EXTERNAL,
FOR EVER BIT(1) STATIC EXTERNAL,
(UNIQ_KEY,NLOCK,SLOCK,XLOCK,SFILE,XFILE,RLOCK) FIXED STATIC EXTERNAL;
```

DATABASE1.PLI

```
DATABASE:
  PROC OPTIONS (MAIN);

/* .....
  DATABASE EXAMPLE  VERSION 1.05  4/20/82  1943
  .....
*/

%INCLUDE 'DATABASE.DCL';

/*
  INTERFACE TO AM80(tm)

  AM80EXTR.PLI CONTAINS THE EXTERNAL DEFINITIONS OF THE AM-80 ROUTINES
*/

%INCLUDE 'AM80EXTR.PLI';

DCL
  ENTDAT ENTRY (CHAR(3),FIXED) RETURNS (CHAR(ACTION_LEN));

/* .....
  SET-UP DATABASE FIELD & KEY DESCRIPTORS
  .....
*/

NO_FIELDS = MAX_FIELD + 1;

FLD_NAME(0) = 'Customer Number';
FLD_LEN(0) = 4;
FLD_NAME(1) = 'First Name';
FLD_LEN(1) = 16;
FLD_NAME(2) = 'Last Name';
FLD_LEN(2) = 20;
FLD_NAME(3) = 'Street Address';
FLD_LEN(3) = 20;
FLD_NAME(4) = 'City';
FLD_LEN(4) = 20;
FLD_NAME(5) = 'State';
FLD_LEN(5) = 2;
```

**Listing 3-2. (continued)**

```

FLD_NAME(6) = 'Zipcode';
FLD_LEN(6) = 9;
FLD_NAME(7) = 'Customer Status';
FLD_LEN(7) = 8;

DCL
  DATBUF_PTR POINTER,
  1 CUST_REC,
    2 CDF CHAR(1),
    2 CNO CHAR(4),
    2 CFN CHAR(16),
    2 CLN CHAR(20),
    2 CST CHAR(20),
    2 CTY CHAR(20),
    2 CSA CHAR(2),
    2 CZP CHAR(9),
    2 CSU CHAR(8);

KEY_LEN(0)=10;
KEY_TYPE(0)=0;
KEY_MAP(0)=2 ; /* KEY 0 = LAST NAME */
KEY_LEN(1)=11;
KEY_TYPE(1)=0;
KEY_MAP(1)=6 ; /* KEY 1 = ZIPCODE */
KEY_LEN(2)=4 ;
KEY_TYPE(2)=0;
KEY_MAP(2)=0 ; /* KEY 2 = CUST NUMBER */
UNIQ_KEY = 2 ; /* USED IN TEST OF UNIQUENESS */

DO KEY = 0 TO MAX_KEY;
  IF KEY = UNIQ_KEY THEN
    KEY_DUP(KEY) = NO;
  ELSE
    KEY_DUP(KEY) = YES;

  KEY_NAME(KEY) = FLD_NAME(KEY_MAP(KEY));
END;

IDX_NAME(0) = 'NAME.IDX';
IDX_NAME(1) = 'ZIPC.IDX';
IDX_NAME(2) = 'NUMB.IDX';

NLOCK = 0; /* IGNORE LOCKS */
SLOCK = 1; /* SHARED RECORD LOCK */
XLOCK = 2; /* EXCLUSIVE RECORD LOCK */
SFILE = 3; /* SHARED FILE LOCK */
XFILE = 4; /* EXCLUSIVE FILE LOCK */
RLOCK = 5; /* RELEASE SLOCK OR XLOCK */

/*
.....

```

Listing 3-2. (continued)



```

INITIALIZE INDEX FILES
*
*****
SET LENGTH = '12345678901';
IDX KEY = SET LENGTH;
SPACE =      '      ';
/*
SET TERMINAL TO -1 FOR AUTOMATIC ASSIGNMENT BY AM-80
*/
*****
TERMINAL = -1;
TRAP_ERRORS = YES;
TIME_OUT_TEST_DELAY = 2; /* APPROXIMATELY 2 SECONDS */
TERMINAL = INTUSR(TERMINAL,TRAP_ERRORS,TIME_OUT_TEST_DELAY);
IF ERRCOD() ~= 0 THEN
    CALL ERROR_TYPE(0,14);

NO_BUFFERS = 5;
NO_NODE_SECTORS = 4;
NO_DATA_FILES = 1;
NO_KEYS = MAX_KEY + 1;

IF SETUP(NO_BUFFERS,NO_KEYS,NO_NODE_SECTORS,NO_DATA_FILES) ~= 0 THEN
    DO;
    PUT SKIP LIST('Illegal SETUP Parameters');
    STOP;
    END;

DO KEY = 0 TO MAX_KEY;
    KEY_NUM(KEY) = OPNIDX(-1,IDX_NAME(KEY),
                        KEY_LEN(KEY), KEY_TYPE(KEY),KEY_DUP(KEY));
    IF ERRCOD() ~= 0 THEN
        CALL ERROR_TYPE(KEY,1);
END;
/*
*****
INITIALIZE DATA FILE
*****
*/
FILE NO = -1;
RECORD_LENGTH = 100;
FILENAME = 'CUSTOMER.DAT';
FILE NO = OPNDAT(FILE_NO,SFILE,FILNAME,RECORD_LENGTH);
IF ERRCOD() ~= 0 THEN
    CALL ERROR_TYPE(0,10);
IF LOKCOD() ~= 0 THEN
    CALL LOKTYP(1);

```

**Listing 3-2. (continued)**

```

/*      CUST_REC IS THE DATA FILE BUFFER AREA
*/

      DATBUF_PTR = ADDR(CUST_REC);

/*
.....
      BEGIN DATABASE OPERATION
.....
*/
      OPEN FILE (SYSCON) OUTPUT TITLE('$CON');
      OPEN FILE (SYSLST) OUTPUT TITLE('$LST');

      FOR EVER = YESBIT;
      DO WHILE (FOR EVER);
          CALL DATA_BASE();
      END;

DATA_BASE:
      PROC;
DCL
      (LOCK_CODE,NDRN,DRN,CHOICE) FIXED,
      (SAVE_KEY,LDRN,NO_LISTED) FIXED,
      ROUTE CHAR(1),
      (CONTINUE,STAYPUT) BIT(1),
      (L_VALUE,U_VALUE,CONV_TARGET,TARGET) CHAR(MAX_KEY_LEN) VAR,
      ACTION CHAR(ACTION_LEN);

      CALL CLRSCR();
      CHOICE = MAIN_MENU();
      GOTO DB(CHOICE);

/*
.....
      ENTER NEW CUSTOMERS
.....
*/
DB(1):
      ACTION = ENTDAT('NEW',0);
      LOCK_CODE = 0;
      IF ACTION = 'SAVE' THEN
          DO;
              NDRN = UPDATE(0);
              LOCK_CODE = FRELOK(FILE_NO,XLOCK,NDRN);
          END;

      IF LOCK_CODE ^= 0 THEN
          CALL LOKTYP(8);

```

**Listing 3-2. (continued)**

```

IF ACTION = 'SAVE' THEN
    GOTO DB(1);
ELSE
    RETURN;

/*
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
SCAN/UPDATE/DELETE CUSTOMERS
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
*/
DB(2):
KEY = SEARCH_KEY();
PUT SKIP(2) EDIT ('Enter target value for ',KEY_NAME(KEY),'',
                ' or enter a period (.) to see main menu>>')
                (3A,SKIP,A);
GET LIST (TARGET);
IF TARGET = '.' THEN RETURN;

CONV_TARGET = KEY_FORMAT(KEY,TARGET);
STAYPUT = YESBIT;
DO WHILE (STAYPUT);
    DRN = SBRKEY(KEY_NUM(KEY),FILE_NO,SLOCK,
                CONV_TARGET,IDX_KEY);
    IF ERRCOD() = 0 THEN
        CALL ERROR_TYPE(KEY,2);
    IF LOKCOD() = 0 THEN
        STAYPUT = CHECK_LOCK(KEY,DRN);
    ELSE
        STAYPUT =NOBIT;
END;

OLD_ACTION = 'CONT';
CONTINUE = YESBIT;
DO WHILE (CONTINUE & DRN = 0);
    LDRN = DRN;
    CALL READ_CUST(DRN);
    ACTION = ENTDAT('OLD',DRN);
    SAVE_KEY = KEY;
    IF ACTION = 'SAVE' THEN
        DRN = UPDATE(DRN);
    IF ACTION = 'DELT' THEN
        CALL DELETE(DRN);
    IF ACTION = 'DELT' & FRELOK(FILE_NO,RLOCK,LDRN) = 0
        THEN CALL LOKTYP(2);
    IF ACTION = 'SAVE' | ACTION = 'DELT' THEN
        DO;
        KEY = SAVE_KEY;
        ACTION = OLD_ACTION;
        END;
    OLD_ACTION = ACTION;
    CONV_TARGET = SUBSTR(IDX_KEY,1,KEY_LEN(KEY));

```

Listing 3-2. (continued)

```

IDX_KEY = SET_LENGTH;
LOCK_CODE = 0;

STAYPUT = YESBIT;
DO WHILE (STAYPUT);
  IF ACTION = 'CONT' THEN
    DO;
    DRN = AFTKEY(KEY_NUM(KEY),FILE_NO,
                SLOCK, CONV_TARGET,IDX_KEY);
    LOCK_CODE = LOKCOD();
    END;
  IF ACTION = 'BACK' THEN
    DO;
    DRN = BEFKEY(KEY_NUM(KEY),FILE_NO,
                SLOCK, CONV_TARGET,IDX_KEY);
    LOCK_CODE = LOKCOD();
    END;

  IF LOCK_CODE ^= 0 THEN
    STAYPUT = CHECK_LOCK(KEY,DRN);
  ELSE
    STAYPUT = NOBIT;
END;

IF ACTION = 'STOP' THEN
  CONTINUE = NOBIT;
END;

PUT SKIP(2) LIST ('SCAN ENDED');
CALL PAUSE();
RETURN;

/*
:.....:
LIST CUSTOMERS
:.....:
*/

DB(3):
KEY = SEARCH_KEY();
PUT SKIP(2) LIST (
'Do you want listing routed to printer (Y/N) >>');
GET LIST (ROUTE);
IF ROUTE = 'y' THEN ROUTE = 'Y';

PUT SKIP(3) EDIT (
'Enter lower and upper limits for ',KEY_NAME(KEY),' listing',
' separate values with a space >>') (3A,SKIP,A);
GET LIST (L_VALUE,U_VALUE);
L_VALUE = KEY_FORMAT(KEY,L_VALUE);
U_VALUE = KEY_FORMAT(KEY,U_VALUE);
DRN = SERKEY(KEY_NUM(KEY),FILE_NO,SLOCK,
            L_VALUE,IDX_KEY);

```

**Listing 3-2. (continued)**

```

IF LOKCOD() ^= 0 THEN
  CALL SKIP_LOCK(KEY,DRN);

NO_LISTED = 0;
DO WHILE (DRN ^= 0 & COMPARE(KEY,IDX_KEY,U_VALUE) <= 0);
  CALL READ_CUST(DRN);
  CALL PRINT_CUST(ROUTE);
  NO_LISTED = NO_LISTED + 1;
  IF FRELOK(FILE_NO,SLOCK,DRN) ^= 0 THEN
    CALL LOKTYP(4);
    L_VALUE = SUBSTR(IDX_KEY,1,KEY_LEN(KEY));
    IDX_KEY = SET_LENGTH;
    DRN = AFTKEY(KEY_NUM(KEY),FILE_NO,SLOCK,
      L_VALUE,IDX_KEY);
    IF LOKCOD() ^= 0 THEN
      CALL SKIP_LOCK(KEY,DRN);
  END;

IF DRN ^= 0 THEN
  LOCK_CODE = FRELOK(FILE_NO,SLOCK,DRN);
ELSE
  LOCK_CODE = 0;
IF LOCK_CODE ^= 0 THEN
  CALL LOKTYP(5);

PUT SKIP(2) EDIT (NO_LISTED,' records listed.') (F(6),A);
CALL PAUSE();
RETURN;

/*
.....

      DATABASE STATISTICS

.....
*/

DB(4):
CALL CLRSCR();
PUT SKIP EDIT (FILENAME,' has ',GETDFS(FILE_NO),
  ' records; currently, ',GETDFU(FILE_NO),
  ' of them are in use.') (2A,F(6),A,F(6),A);
PUT SKIP(4) EDIT ('      INDEX','ENTRIES') (A,COLUMN(30),A);
PUT SKIP EDIT ('-----','-----') (A,COLUMN(30),A);
DO KEY = 0 TO MAX_KEY;
  PUT SKIP EDIT (KEY_NAME(KEY),NOKEYS(KEY)) (A,COLUMN(30),F(6));
END;

PUT SKIP(4);
CALL PAUSE();
RETURN;

/*
.....

```

Listing 3-2. (continued)

```

                SAVE DATABASE UPDATES & RESTART
*/
:
DB(5):
  IF SAVDAT(FILE_NO) ^= 0 THEN
    CALL ERROR_TYPE(0,7);
  DO KEY = 0 TO MAX KEY;
    IF SAVIDX(KEY_NUM(KEY)) ^= 0 THEN
      CALL ERROR_TYPE(KEY,3);
  END;
  RETURN;
/*
:
                SAVE DATABASE UPDATES & TERMINATE
*/
:
DB(6):
  CLOSE FILE (SYSLST);
  IF CLSDAT(FILE_NO) ^= 0 THEN
    CALL ERROR_TYPE(0,15);
  DO KEY = 0 TO MAX KEY;
    IF CLSIDX(KEY_NUM(KEY)) ^= 0 THEN
      CALL ERROR_TYPE(KEY,16);
  END;
  IF FRELOK(FILE_NO,SFILE,0) ^= 0 THEN
    CALL ERROR_TYPE(0,13);
  PUT SKIP(2) LIST (' *** SUCCESSFUL TERMINATION ***');
  STOP;
END DATA_BASE;
/*
+++++
                BEGINNING OF UTILITY FUNCTIONS
:
                CLEAR SCREEN ROUTINE
:
*/
CLRSCR:
  PROC EXTERNAL;
DCL

```

**Listing 3-2. (continued)**

```

        DUMMY FIXED BINARY(7);
        DO DUMMY = 1 TO 24;
            PUT SKIP;
        END;
    END CLRSCR;

/*
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        MAIN MENU ROUTINE
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
*/
MAIN_MENU:
    PROC RETURNS (FIXED);
DCL
    OP FIXED;

    PUT SKIP EDIT (' AM-80(tm) DEMONSTRATION') (X(20),A);
    PUT SKIP(2) EDIT(' Customer Database Operations') (X(20),A);
    PUT SKIP EDIT(' Terminal ',TERMINAL) (X(20),A,F(2));
    PUT SKIP EDIT(' *****') (X(20),A);
    PUT SKIP(3) EDIT('1. Enter New Customers') (X(5),A);
    PUT SKIP EDIT('2. Scan/Update/Delete Customer Records') (X(5),A);
    PUT SKIP EDIT('3. List Customer Records') (X(5),A);
    PUT SKIP EDIT('4. Database Statistics') (X(5),A);
    PUT SKIP EDIT('5. Save All Files & Restart Operations') (X(5),A);
    PUT SKIP EDIT('6. Terminate Operations') (X(5),A);

    OP = 0;
    DO WHILE (OP < 1 | OP > 6);
        PUT SKIP(2) LIST ('Enter desired operation number>>');
        GET LIST (OP);

    END;
    RETURN(OP);
END MAIN_MENU;

/*
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        SELECT SEARCH KEY ROUTINE
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
*/
SEARCH_KEY:
    PROC RETURNS (FIXED);
DCL
    (KEY,KEY_NO) FIXED;

    CALL CLRSCR();
    PUT EDIT ('Customer Database Search Keys') (X(25),A);
    PUT SKIP (3);

```

Listing 3-2. (continued)

```

DO KEY = 0 TO MAX KEY;
  KEY_NO = KEY + 1;
  PUT SKIP EDIT(KEY_NO,' - ',KEY_NAME(KEY)) (X(5),F(3),2A);
END;

KEY = 0;
DO WHILE (KEY < 1 | KEY > NO_KEYS);
  PUT SKIP(3) LIST('Enter desired key number>>');
  GET LIST (KEY);
END;
RETURN(KEY-1);
END SEARCH_KEY;

/*
.....
          ERROR HANDLING
.....
*/

ERROR_TYPE:
  PROC (INFO,TYPE);

DCL
  (T_KEY,INFO,DUMMY,TYPE) FIXED;

  PUT SKIP(3) EDIT ('User Error #',ERRCOD(),' occurred while trying to ')
    (A,F(4),A);

  GOTO ET(TYPE);

ET(1):  PUT EDIT ('open ',IDX_NAME(INFO)) (2A);
        GOTO ET_STOP;
ET(2):  PUT EDIT ('search ',KEY_NAME(INFO),' Index File') (3A);
        GOTO ET_CLOSE;
ET(3):  PUT EDIT('save ',IDX_NAME(INFO)) (2A);
        GOTO ET_PCLOSE;
ET(4):  PUT EDIT('remove old key from ',IDX_NAME(INFO)) (2A);
        GOTO ET_CLOSE;
ET(5):  PUT EDIT('enter key into ',IDX_NAME(INFO)) (2A);
        GOTO ET_CLOSE;
ET(6):  PUT EDIT('delete key from ',IDX_NAME(INFO)) (2A);
        GOTO ET_CLOSE;
ET(7):  PUT EDIT('save ',FILNAME) (2A);
        INFO = -1;
        GOTO ET_PCLOSE;
ET(8):  PUT EDIT('get a new data record',' (' ,FILE_NO,')') (2A,F(3),A);
        GOTO ET_STOP;
ET(9):  PUT EDIT('delete data record #',INFO) (A,F(6));
        GOTO ET_STOP;
ET(10): PUT EDIT('open ',FILNAME,' (' ,FILE_NO,')') (2A,F(3),A);
        GOTO ET_STOP;
ET(11): PUT EDIT('read data record #',INFO) (A,F(6));
        GOTO ET_STOP;

```

**Listing 3-2. (continued)**



```

ET(12): PUT EDIT('write data record #',INFO) (A,F(6));
        GOTO ET_STOP;
ET(13): PUT EDIT('release shared file lock on ',FILNAME) (2A);
        GOTO ET_STOP;
ET(14): PUT EDIT('initialize user.') (A);
        STOP;
ET(15): PUT EDIT('close ',FILNAME) (2A);
        INFO = -1;
        GOTO ET_PCLOSE;
ET(16): PUT EDIT('close ',IDX_NAME(INFO)) (2A);
        GOTO ET_PCLOSE;

ET_CLOSE:
        DUMMY = CLSDAT(FILE_NO);
        DO T_KEY = 0 TO MAX_KEY;
                IF T_KEY ^= INFO THEN DUMMY = CLSIDX(KEY_NUM(T_KEY));
        END;
        GOTO ET_STOP;

ET_PCLOSE:
        T_KEY = INFO + 1;
        IF T_KEY > MAX_KEY THEN STOP;
        DO INFO = T_KEY TO MAX_KEY;
                DUMMY = CLSIDX(KEY_NUM(INFO));
        END;

ET_STOP:
        PUT SKIP(2) EDIT('DEMONSTRATION TERMINATING WITH ERROR CODE #',
                ERRCOD()) (A,F(4));
        STOP;

END ERROR_TYPE;

LOKTYP:
        PROC (TYPE) EXTERNAL;
DCL
        (T_KEY,DUMMY,TYPE) FIXED;

        PUT SKIP EDIT('Lock Type: ',TYPE,' Lock Code:',LOKCOD())
                (A,F(3),A,F(3));
        DUMMY = CLSDAT(FILE_NO);
        DO T_KEY = 0 TO MAX_KEY;
                DUMMY = CLSIDX(KEY_NUM(T_KEY));
        END;
        STOP;
END LOKTYP;

/*
        .....
                STRIP TRAILING BLANKS
        .....
*/

```

Listing 3-2. (continued)

```

STRIP_BLANKS:
  PROC (OLD_VAL) RETURNS (CHAR(MAX_FLD_LEN) VAR);
DCL
  OLD_VAL CHAR(MAX_FLD_LEN),
  (TEST,FLDLN) FIXED;

  FLDLEN = LENGTH(OLD_VAL);
  DO TEST = FLDLEN TO 1 BY -1;
    IF SUBSTR(OLD_VAL,TEST,1) ~= ' ' THEN
      RETURN (SUBSTR(OLD_VAL,1,TEST));
    END;
  RETURN ('');
END STRIP_BLANKS;

/*
  ::::::::::::::::::::::::::::::::::::::::::::::::::::
  READ DATA RECORD ROUTINE
  ::::::::::::::::::::::::::::::::::::::::::::::::::::
*/
READ_CUST:
  PROC (DRN);
DCL
  DRN FIXED;

  IF READAT(FILE_NO,DRN,DATBUF_PTR) ~= 0 THEN
    CALL ERROR_TYPE(DRN,11);

  OLD_FLD(0) = STRIP_BLANKS(CNO);
  OLD_FLD(1) = STRIP_BLANKS(CFN);
  OLD_FLD(2) = STRIP_BLANKS(CLN);
  OLD_FLD(3) = STRIP_BLANKS(CST);
  OLD_FLD(4) = STRIP_BLANKS(CTY);
  OLD_FLD(5) = STRIP_BLANKS(CSA);
  OLD_FLD(6) = STRIP_BLANKS(CZF);
  OLD_FLD(7) = STRIP_BLANKS(CSU);
END READ_CUST;

/*
  ::::::::::::::::::::::::::::::::::::::::::::::::::::
  LIST CUSTOMER RECORD ROUTINE
  ::::::::::::::::::::::::::::::::::::::::::::::::::::
*/
PRINT_CUST:
  PROC (ROUTE);
DCL
  ROUTE CHAR(1),
  LIST_FILE FILE VARIABLE;

LFRMT:

```

Listing 3-2. (continued)

```

FORMAT(X(24),5A);

IF ROUTE = 'Y' THEN
    LIST_FILE = SYSLST;
ELSE
    LIST_FILE = SYSCON;

PUT FILE(LIST_FILE) SKIP (2) EDIT(OLD_FLD(0),OLD_FLD(7))
    (X(4),A,COLUMN(15),A);
PUT FILE(LIST_FILE) EDIT(OLD_FLD(1),' ',OLD_FLD(2)) (R(LFRMT));
PUT FILE(LIST_FILE) EDIT(OLD_FLD(3)) (R(LFRMT));
PUT FILE(LIST_FILE) EDIT(OLD_FLD(4),' ',OLD_FLD(5),' ',OLD_FLD(6))
    (R(LFRMT));
PUT FILE(LIST_FILE) SKIP;
END PRINT_CUST;

/*
:
:
:           PAUSE ROUTINE
:
:
*/
PAUSE:
PROC;
DCL
    DUMMY CHAR(1);

    PUT SKIP(2) LIST ('Enter any character to continue ---');
    GET LIST (DUMMY);
END PAUSE;

/*
:
:
:   CONVERT TARGET VALUE TO KEY FORMAT ROUTINE
:
:
*/
KEY_FORMAT:
PROC (KEY,TARGET) RETURNS (CHAR(MAX_KEY_LEN) VAR);
DCL
    KEY FIXED,
    TEMP CHAR(40) VAR,
    TARGET CHAR(MAX_KEY_LEN) VAR;

    IF UNIQ_KEY = KEY THEN
        RETURN(TARGET);
    ELSE
        DO;
            TEMP = TARGET || SPACE;
            RETURN(SUBSTR(TEMP,1,KEY_LEN(KEY)-2) ||

```

**Listing 3-2. (continued)**

```

                ASCII(0) || ASCII(0));
END;
END KEY_FORMAT;

/*
.....
                COMPARE IDX_KEY & U_VALUE ROUTINE
.....
*/

COMPARE:
PROC (KEY,IDXVAL,UPVAL) RETURNS (FIXED);
DCL
(KL,KEY) FIXED,
(C1,C2) CHAR(40) VAR,
(IDXVAL,UPVAL) CHAR(MAX_KEY_LEN) VAR;

IF KEY = UNIQ_KEY THEN
    KL = KEY_LEN(KEY);
ELSE
    KL = KEY_LEN(KEY)-2;

C1 = IDXVAL || SPACE;
C1 = SUBSTR(C1,1,KL);
C2 = UPVAL || SPACE;
C2 = SUBSTR(C2,1,KL);

IF C1<C2 THEN
    RETURN(-1);
ELSE IF C1>C2 THEN
    RETURN(1);
ELSE
    RETURN(0);
END COMPARE;

/*
.....
                CHECK LOCK ROUTINES
.....
*/

SKIP_LOCK:
PROC (KEY,DRN);
DCL
L_VALUE CHAR(MAX_KEY_LEN) VAR,
(KEY,DRN) FIXED;

DO WHILE (DRN ^= 0 & LOKCOD() ^= 0);
    L_VALUE = SUBSTR(IDX_KEY,1,KEY_LEN(KEY));
    IDX_KEY = SET_LENGTH;
    DRN = APTKEY(KEY_NUM(KEY),FILE_NO,SLOCK,

```

**Listing 3-2. (continued)**

```

                                L_VALUE,IDX_KEY);
END;
END SKIP_LOCK;

CHECK_LOCK:
PROC (KEY,DRN) RETURNS (BIT(1));
DCL
CONV_TARGET CHAR(MAX_KEY_LEN) VAR,
(KEY,DRN) FIXED,
DUMMY CHAR(1);

PUT SKIP(2) LIST(
'Enter a "w" if you want to wait for locked record(s)>>');
GET LIST (DUMMY);

IF DUMMY = 'W' | DUMMY = 'w' THEN
RETURN(YESBIT);

DO WHILE (DRN ^= 0 & LOKCOD() ^= 0);
CONV_TARGET = SUBSTR(IDX_KEY,1,KEY_LEN(KEY));
IDX_KEY = SET_LENGTH;
IF OLD_ACTION = 'CONT' THEN
DRN = APTKEY(KEY_NUM(KEY),FILE_NO,
SLOCK, CONV_TARGET,IDX_KEY);
ELSE
DRN = BEFKEY(KEY_NUM(KEY),FILE_NO,
SLOCK, CONV_TARGET,IDX_KEY);
END;
RETURN(NOBIT);
END CHECK_LOCK;

/*
.....
WARNING MESSAGES
.....
*/

WARNING_TYPE:
PROC (KEY,TYPE,RET_CODE);
DCL
(KEY,TYPE,RET_CODE) FIXED;

PUT SKIP(2) EDIT ('WARNING...Return Code #',RET_CODE,
' occurred while trying to ') (A,F(3),A);
GOTO WT(TYPE);

WT(1): PUT EDIT ('remove old key from ',IDX_NAME(KEY)) (2A);
CALL PAUSE();
RETURN;

WT(2): PUT EDIT ('enter key into ',IDX_NAME(KEY)) (2A);
CALL PAUSE();
RETURN;

```

**Listing 3-2. (continued)**

```

WT(3): PUT EDIT ('delete key from ',IDX_NAME(KEY)) (2A);
      CALL PAUSE();
      RETURN;

END WARNING_TYPE;

/*
      .....

      ADD NEW KEY VALUE ROUTINE

      .....
*/

ADD_KEY:
PROC (KEY,DRN);
DCL
  (KEY,RET_CODE,K_FLD,DRN) FIXED;

  K_FLD = KEY_MAP(KEY);

/*
  REMOVE OLD KEY VALUE
*/

  RET_CODE = DELKEY(KEY_NUM(KEY),FILE_NO,
    XLOCK,OLD_FLD(K_FLD),DRN);

  IF ERRCOD() ~= 0 THEN
    CALL ERROR_TYPE(KEY,4);
  IF LOKCOD() ~= 0 THEN
    CALL LOKTYP(6);
  IF RET_CODE ~= 1 THEN
    CALL WARNING_TYPE(KEY,1,RET_CODE);

/*
  ADD NEW KEY VALUE
*/

  RET_CODE = ADDKEY(KEY_NUM(KEY),FILE_NO,
    XLOCK,NEW_FLD(K_FLD),DRN);

  IF ERRCOD() ~= 0 THEN
    CALL ERROR_TYPE(KEY,5);
  IF LOKCOD() ~= 0 THEN
    CALL LOKTYP(7);
  IF RET_CODE ~= 1 THEN
    CALL WARNING_TYPE(KEY,2,RET_CODE);

END ADD_KEY;

/*
      .....

```

**Listing 3-2. (continued)**

```

WRITE NEW DATA RECORD ROUTINE
/*
WRITE_CUST:
  PROC (DRN);
DCL
  DRN FIXED;

  CDF = ASCII(0);          /* CLEAR DELETE FLAG */
  CNO = NEW_FLD(0);
  CFN = NEW_FLD(1);
  CLN = NEW_FLD(2);
  CST = NEW_FLD(3);
  CTY = NEW_FLD(4);
  CSA = NEW_FLD(5);
  CZP = NEW_FLD(6);
  CSU = NEW_FLD(7);

  IF WRDAT(FILE_NO,DRN,DATBUF_PTR) ^= 0 THEN
    CALL ERROR_TYPE(DRN,I2);
END WRITE_CUST;
/*
DELETE KEY VALUE FROM INDEX ROUTINE
/*
DEL_KEY:
  PROC (KEY,DRN);
DCL
  (KEY,RET_CODE,K_FLD,DRN);

  K_FLD = KEY_MAP(KEY);

  RET_CODE = DELKEY(KEY_NUM(KEY),FILE_NO,
    XLOCK,OLD_FLD(K_FLD),DRN);

  IF ERRCOD() ^= 0 THEN
    CALL ERROR_TYPE(KEY,6);
  IF LOKCOD() ^= 0 THEN
    CALL LOKTYP(10);
  IF RET_CODE ^= 1 THEN
    CALL WARNING_TYPE(KEY,3,RET_CODE);
END DEL_KEY;
/*
UPDATE INDICES & DATA FILE ROUTINE

```

## Listing 3-2. (continued)

```

/*
*****
UPDATE:
PROC (DATA_RECORD) RETURNS (FIXED);
DCL
  (FLD,KEY) FIXED,
  (TMP_REC,DATA_RECORD) FIXED;
  IF DATA_RECORD = 0 THEN
    DO;
      TMP_REC = NEWREC (FILE_NO,XLOCK);
      IF ERRCOD() ^= 0 THEN
        CALL ERROR_TYPE(0,8);
      IF LOKCOD() ^= 0 THEN
        CALL LOKTYP(3) ;
      END;
    ELSE
      TMP_REC = DATA_RECORD;
  DO KEY = 0 TO MAX_KEY;
    FLD = KEY_MAP(KEY);
    IF OLD_FLD(FLD) ^= NEW_FLD(FLD) THEN
      CALL ADD_KEY(KEY,TMP_REC);
  END;
  DO FLD = 0 TO MAX_FIELD;
    IF OLD_FLD(FLD) ^= NEW_FLD(FLD) THEN
      DO;
        CALL WRITE_CUST(TMP_REC);
        RETURN(TMP_REC);
      END;
  END;
  RETURN(TMP_REC);
END UPDATE;
/*
*****
DELETE INDEX & DATA FILE ENTRY ROUTINE
*****
*/
DELETE:
PROC (DATA_RECORD);
DCL
  (DATA_RECORD,FLD,KEY) FIXED;
  DO KEY = 0 TO MAX_KEY;
    FLD = KEY_MAP(KEY);
    IF OLD_FLD(FLD) ^= '' THEN
      CALL DEL_KEY(KEY,DATA_RECORD);
  END;

```

Listing 3-2. (continued)



```

        IF RETREC(FILE NO,XLOCK,DATA_RECORD) ^= 0 THEN
            CALL ERROR_TYPE(DATA_RECORD,9);
        IF LOKCOD() ^= 0 THEN
            CALL LOKTYP(9);
    END DELETE;

/*
        END OF UTILITY FUNCTIONS

+++++
*/
END;
```

## DATABAS2.PLI

```

ENTDAT:
    PROC (ENTER_MODE,DRN) RETURNS (CHAR(4)) EXTERNAL;

/*
    :
    DATA ENTRY ROUTINE
    :
*/

DCL
    UNIQUE BIT(1),
    (DRN,FLD,OP_VAL,FLD_NO) FIXED,
    OP CHAR(2) VAR,
    OP1 CHAR(1),
    TEMP_MODE FIXED,
    ENTER_MODE CHAR(3);

%INCLUDE 'DATABASE.DCL';
%INCLUDE 'AM80EXTR.PLI';

DCL
    CLRSCR ENTRY,
    LOKTYP ENTRY (FIXED);

    IF ENTER_MODE = 'NEW' THEN
        DO FLD = 0 TO MAX_FIELD;
            OLD_FLD(FLD) = '';
        END;

    IF ENTER_MODE = 'OLD' THEN
        DO FLD = 0 TO MAX_FIELD;
```

**Listing 3-2. (continued)**

```

        NEW_FLD(FLD) = OLD_FLD(FLD);
    END;

CALL CLRSCR();

IF ENTER_MODE = 'NEW' THEN
    DO;
    PUT SKIP EDIT ('Enter New Customer Information') (X(19),A);
    PUT SKIP EDIT ('*****') (X(19),A);
    PUT SKIP(3) LIST (
        '      [Enter zero for customer # to see main menu.]);
    PUT SKIP(2);

    DO FLD = 0 TO MAX_FIELD;
        FLD_NO = FLD + 1;
REDO_DATA:
        PUT EDIT (FLD_NO, ' - ', FLD_NAME(FLD),
            ' (', FLD_LEN(FLD), ') >>')
            (F(6), 2A, COLUMN(30), A, F(2), A);
        GET LIST (NEW_FLD(FLD));
        IF FLD = KEY_MAP(UNIQ_KEY) & NEW_FLD(FLD) = '0' THEN
            RETURN('STOP');

        IF FLD = KEY_MAP(UNIQ_KEY) THEN
            DO;
            NEW_FLD(FLD) = RIGHT('0000' || NEW_FLD(FLD),
                FLD_LEN(FLD));
            UNIQUE = TEST_UNIQUENESS();
            END;
        ELSE
            DO;
            NEW_FLD(FLD) = SUBSTR(NEW_FLD(FLD), 1,
                FLD_LEN(FLD));
            UNIQUE = YESBIT;
            END;

        IF ~UNIQUE THEN GOTO REDO_DATA;
    END;
    TEMP_MODE = NEW_MODE;
END;
ELSE
    TEMP_MODE = OLD_MODE;

DO WHILE (FOR_EVER);
    PUT SKIP(4) EDIT ('Current customer information')
        (X(19),A);
    PUT SKIP;

    DO FLD = 0 TO MAX_FIELD;
        FLD_NO = FLD + 1;
        PUT SKIP EDIT (FLD_NO, ' - ', FLD_NAME(FLD), NEW_FLD(FLD))
            (F(6), 2A, COLUMN(30), A);
    END;

    IF TEMP_MODE = OLD_MODE THEN

```

**Listing 3-2. (continued)**

```

        BEGIN;
        OP_VAL = 0;
        ON ERROR(1)
            BEGIN;
            OP_VAL = 0;
            GOTO RETRY_OLD;
            END;
RETRY_OLD:
        DO WHILE (OP_VAL < 1 | OP_VAL > NO_FIELDS);
            PUT SKIP(3) EDIT (
'Enter C to continue scan, Field # to change data, S to save changes,',
'D to delete data, B for back scan, or E to end scan >>') (A,SKIP,A);
            GET LIST (OP);
            OP1 = OP;
            IF OP1 = 'C' | OP1 = 'c' THEN RETURN('CONT');
            IF OP1 = 'S' | OP1 = 's' THEN
                RETURN(SET_XLOCK(OP1,DRN));
            IF OP1 = 'D' | OP1 = 'd' THEN
                RETURN(SET_XLOCK(OP1,DRN));
            IF OP1 = 'B' | OP1 = 'b' THEN RETURN('BACK');
            IF OP1 = 'E' | OP1 = 'e' THEN RETURN('STOP');
            OP_VAL = OP;
        END;
        CALL UPDATE_FIELD(OP_VAL);
        END;
    ELSE
        BEGIN;
        OP_VAL = 0;
        ON ERROR(1)
            BEGIN;
            OP_VAL = 0;
            GOTO RETRY_NEW;
            END;
RETRY_NEW:
        DO WHILE (OP_VAL < 1 | OP_VAL > NO_FIELDS);
            PUT SKIP (3) EDIT (
'Enter S to save data, Field # to change data,',
'D to delete data, or E to end input >>') (A,SKIP,A);
            GET LIST (OP);
            OP1 = OP;
            IF OP1 = 'S' | OP1 = 's' THEN RETURN('SAVE');
            IF OP1 = 'D' | OP1 = 'd' THEN RETURN('DELT');
            IF OP1 = 'E' | OP1 = 'e' THEN RETURN('STOP');
            OP_VAL = OP;
        END;
        CALL UPDATE_FIELD(OP_VAL);
        END;
    END;

SET_XLOCK:
    PROC (OP,DRN) RETURNS (CHAR(ACTION_LEN));
DCL
    DRN FIXED,
    (DUMMY,OP) CHAR(1);

```

**Listing 3-2. (continued)**

```

DUMMY = 'W';
DO WHILE (DUMMY = 'W' & SETLOK(FILE_NO,XLOCK,DRN) ^= 0);
  PUT SKIP (2) EDIT (
'Customer update on hold due to record lock',
'Enter W if you want to wait or any other key to cancel update>>')
(A,SKIP,A);
  GET LIST (DUMMY);
  IF DUMMY = 'w' THEN DUMMY = 'W';
END;

IF DUMMY = 'W' THEN
DO;
  IF OP = 'S' THEN
    RETURN('SAVE');
  ELSE
    RETURN('DELT');
  END;
ELSE
  RETURN(OLD_ACTION);
END SET_XLOCK;

/*
.....
UPDATE DATA FIELD ROUTINE
.....
*/

UPDATE_FIELD:
PROC (FLD_NO);
DCL
  TEST BIT(1),
  (FLD_NO,FIELD_NO) FIXED;

  FLD_NO = FLD_NO-1;
  TEST = NOBIT;

  DO WHILE (~TEST);
    PUT SKIP(2) EDIT ('Input new ',FLD_NAME(FIELD_NO),'>>')
      (3A);
    GET LIST (NEW_FLD(FIELD_NO));

    IF FIELD_NO = KEY_MAP(UNIQ_KEY) THEN
      NEW_FLD(FIELD_NO) = RIGHT('000Q' || NEW_FLD(FIELD_NO),
        FLD_LEN(FIELD_NO));
    ELSE
      NEW_FLD(FIELD_NO) = SUBSTR(NEW_FLD(FIELD_NO),1,
        FLD_LEN(FIELD_NO));

    IF FIELD_NO = KEY_MAP(UNIQ_KEY) & NEW_FLD(FIELD_NO) ^=
      OLD_FLD(FIELD_NO) THEN
      TEST = TEST_UNIQUENESS();
  
```

Listing 3-2. (continued)

```

                ELSE
                    TEST = YESBIT;
            END;
END UPDATE_FIELD;

/*
.....
                CUST # UNIQUENESS TEST ROUTINE
.....
*/

TEST_UNIQUENESS:
    PROC RETURNS (BIT(1));
DCL
    TEMP FIXED,
    TEST CHAR(MAX_FLD_LEN) VAR;

    TEST = NEW_FLD(KEY_MAP(UNIQ_KEY));
    TEMP = GETKEY(UNIQ_KEY,0,NLOCK,TEST);

    IF LORCOD() ^= 0 THEN
        CALL LOKTYP(12);
    IF TEMP = 0 THEN
        RETURN(YESBIT);
    ELSE
        DO;
            PUT SKIP(2) LIST (' *** Already Assigned ***');
            PUT SKIP;
            RETURN(NOBIT);
        END;
END TEST_UNIQUENESS;

/*
.....
                RIGHT STRING ROUTINE
.....
*/

RIGHT:
    PROC (FLDSTR,FLDLEN) RETURNS (CHAR(MAX_FLD_LEN) VAR);
DCL
    FLDLEN FIXED,
    FLDSTR CHAR(MAX_FLD_LEN) VAR;

    RETURN(SUBSTR(FLDSTR,LENGTH(FLDSTR)-FLDLEN+1));
END RIGHT;

END ENTDAT;

```

**Listing 3-2. (continued)**

End of Section 3

## Section 4

# Using Access Manager with Pascal/MT+ Applications

This section contains instructions for implementing Access Manager with application programs coded in Pascal/MT+.

Two examples are provided. The first illustrates the use of many Access Manager functions described in your Reference Manual, and in particular, how to use the data file functions in your Pascal/MT+ applications. The second example illustrates the use of Access Manager to create and maintain a data base.

### 4.1 Linking Access Manager to Your Application Program

This section discusses a Pascal/MT+ application program called MYPROG that you write and compile to produce a binary relocatable file.

#### 4.1.1 Linking Single-user Pascal/MT+ Applications

You must link your compiled application program to the appropriate Access Manager subroutine library and index file buffer module. You can use the following command line to create an executable version of MYPROG:

```
LINKMT MYPROG,AM80PASC/S,AM80BUF,PASLIB/S
```

AM80BUF contains the buffer area beginning with entry point AM8FCB and ending with AM8END.

Before linking, be sure AM80BUF.ERL is large enough to contain your buffers (as specified in the SETUP function). You can use SETAMBUF to create a correctly sized buffer module.

If LINKMT.COM returns an out-of-memory message, you must modify the link statement so the data segment is explicitly placed after the code segment. The /D switch of LINKMT does this modification. To determine the appropriate origin for the data segment, perform a trial link in which the data segment origin is very high. For example,

```
LINKMT MYPROG,AM80PASC/S,AM80BUF,PASLIB/S/D:C000
```

Then, based on the size of the code segment returned by LINKMT, you can reset the origin of the data segment to a value somewhat above the code segment and perform the final link. For more details, see the example link statements for DATABASE.SRC and RECREATE.SRC later in this section.

### 4.1.2 Linking Multiuser Pascal/MT+ Applications

If your single-user version of MYPROG is coded with appropriate data locking procedures, you do not have to recompile it to create a multiuser version. All that is necessary is to relink the program.

You must link your compiled application program to the appropriate Access Manager multiuser interface. The interface makes the queue calls to the shared code in the background server. The background server resides in its own memory segment.

To create a COM file that calls the Access Manager background server, use LINKMT.COM as follows:

```
LINKMT MYPROG,AMQ8PASC,PASLIB/S
```

### 4.2 External Declaration of Access Manager Routines

Pascal/MT+ requires that external routines (those not coded in the program module but referenced by it) be explicitly declared. The file AM80EXTR.PSC contains external function declarations for the entire set of Access Manager routines. Use the Include File compiler toggle of Pascal/MT+ to make these external declarations a part of your application program. For example,

```
{ $I AM80EXTR.PSC }
```

includes the external declarations as required.

All Access Manager string-valued parameters (FILENAME, IDXNAME, KEYVAL, and IDXVAL) must be declared as type STRING. Strings, as compared to character arrays, reserve the leading byte for a length counter Access Manager needs to determine the actual length of a string-valued parameter.

### 4.3 Coding Numeric Key Values

For a general discussion of coding numeric key values, refer to the ADDKEY function description in Section 3 of your Reference Manual.

In a Pascal/MT+ environment, the most straightforward use of numeric keys is with the BCD REAL variables which store numeric quantities with the most significant digits in the first byte position, the least significant digits in the ninth byte, and the sign indicator in the tenth byte. Because the most significant byte comes last, negative quantities are not properly handled and you should avoid them. The BCD REALS provide eighteen digits including four decimal places.

The following declarations and assignments overlay BCD REALS onto the string variables that must be passed to the Access Manager functions.

```

CONST
    KEYLEN BY 10;                (* BCD REAL uses ten bytes *)

TYPE
    BCDOVL = RECORD;
        LEN : BYTE;
        VAL : REAL;              (* use compiler B switch *)

VAR
    KEYVAL,IDXVAL : STRING[KEYLEN];
    BCDINP,BCDOUT : ^BCDOVL;

{$I AM80EXTR.PSC}

BCDINP := ADDR(KEYVAL);          (* overlay bcd on string *)
BCDINP^.LEN := KEYLEN;          (* set length byte of string *)
BCDOUT := ADDR(IDXVAL);
BCDOUT^.LEN := KEYLEN;

```

To pass key values to and from Access Manager, use KEYVAL and IDXVAL, respectively. To manipulate the key values as numeric quantities, use BCDINP^.VAL and BCDOUT^.VAL. For example,

```

BCDINP^.VAL := 123.4567;
DRN := BEFKEY(KEY NO,DFILE,DLOCK,KEYVAL,IDXVAL);
IF (DRN <> 0) OR (DATVAL <> 0) THEN
    WRITELN(BCDOUT^.VAL);

```

prints the numeric value of the index entry that immediately precedes 123.4567, unless no such entry exists.

The space savings for this approach with Pascal/MT+ is only meaningful if numbers with more than ten digits are involved because BCD REALS are forced to use ten bytes, and hence the key length must be set to ten bytes. Note that the same type of overlaying can be accomplished with INTEGER variables. If you overlay integers instead of reals, the key length must be set as necessary (two bytes for regular integers and four bytes for long integers), and then set KEYTYP to one. The key values are treated as signed integers.

#### 4.4 Using the RECREATE.SRC Utility Program

RECREATE.SRC contains the Pascal/MT+ source code for the RECREATE utility program. You can change the source code in whatever way you want. To create RECREATE.COM, compile RECREATE.SRC using MTPPLUS.COM and then link as follows:

```

LINKMT RECREATE,AM80PASC/S,AM80BUF,FPREALS/S,
RANDOMIO/S,PASLIB/S/D:7E00

```



The buffer area for RECREATE is 4,600 bytes based on these parameter values:

- NNSEC% = 4
- NBUFS% = 8
- NDATA% = 1
- NKEYS% = 1

Note that only one data file and one index file are open at the same time while RECREATE is running. Use SETAMBUF to configure AM80BUF.ERL.

Table 4-1 shows the layout and content of records in a Recreate Parameter File. This particular example file can be used to reconstruct DATABASE.SRC (see Listing 4-2).

**Table 4-1. Example Pascal/MT+ Recreate Parameter File**

Record Type	Contents
Header	1 4
Data File	CUSTOMER.DAT
Data File	100 3 0
Index File	NAME.IDX
Index File	10 0 1 1
Index File	Y
Key Part	22 8
Index File	NUMB.IDX
Index File	4 0 0 1
Index File	N
Key Part	2 4
Index File	ZIPC.IDX
Index File	11 0 1 1
Index File	Y
Key Part	84 9

If you want to change the capacities of the RECREATE program (and, hence, its memory requirements), note the following key Pascal constants:

- MAX\_NO KEYS and MAX\_KEY PARTS specify the maximum number of index files associated with a data file and the maximum number of fields comprising a key value, respectively.
- MAX\_SORT is the maximum number of key values that can be buffered by RECREATE.SRC before being sorted and added to the index file being recreated.
- MAX\_SPACE specifies the actual number of bytes available for the buffered key values. Each key value requires one more byte than its key length.

The actual number of buffered key values depends on the key length. For short key lengths, MAX\_SORT will be the limiting factor. For long key lengths, MAX\_SPACE will be the limiting factor.

#### 4.5 Pascal/MT+ Data File Example

The following listing illustrates use of the primary Access Manager functions to update records in a data file:

```
PROGRAM EXAMPLE;
CONST
  MAX_KEY_LEN = 48;
  NAME_LEN = 14;
TYPE
  INVENTORY = RECORD
    PART_NO : ARRAY[1..4] OF CHAR;
    PART_NAME : ARRAY[1..18] OF CHAR;
    PART_QUAN : REAL; (* 10-byte BCD Real => use B switch
                      of PASCAL/MT+ Compiler *)
  END;
(* -----
   Variable Declarations
  ----- *)
VAR
  N_LOCK,S_LOCK,X_LOCK,S_FILE,X_FILE : INTEGER;
  NBUF,NKEYS,NNSEC,NDATF,ERROPT,PROGID,TIMOUT : INTEGER;
  DRN,DRN2,FILE_NO,RECORD_LEN : INTEGER;
  DAT_BUFFER : INVENTORY;
  DATBUF_PTR : ^INVENTORY;
```

#### Listing 4-1. Pascal/MT+ Data File Example

```

(*) -----
      AM80 External Declarations
-----
*)
{ $I AM80EXTR.PSC }

(*) -----
      Exception Processing Routines
-----
*)
PROCEDURE ERROR_HANDLER(LOCALE : INTEGER);
  BEGIN
    WRITELN('ERROR at ',LOCALE,' with code ',ERRCOD);
  END;

PROCEDURE LOCK_CONFLICT(LOCALE : INTEGER);
  BEGIN
    WRITELN('LOCK Conflict at ',LOCALE,' with code ',LORCOD);
  END;

BEGIN

(*) -----
      Lock Parameter Setup
-----
*)
N_LOCK := 0; (* No lock request *)
S_LOCK := 1; (* Shared record lock *)
X_LOCK := 2; (* Exclusive record lock *)
S_FILE := 3; (* Shared file lock *)
X_FILE := 4; (* Exclusive file lock *)

(*) -----
      System Initialization Parameters
-----
*)
NBUF := 3; (* 3 buffers *)
NKEYS := 1; (* 1 index file *)
NNSEC := 4; (* 512-byte index file record length *)
NDATF := 1; (* 1 data file *)
ERROPT := 1; (* Trap user errors *)
PROGID := -1; (* Program ID assigned to MP/M console no. *)
TIMEOUT := 3; (* Background server time-out delay *)

```

Listing 4-1. (continued)

```
(* -----  
      Initialize System  
-----*)  
*)  
PROGID := INTUSR(PROGID,ERROPT,TIMOUT);  
IF ERRCOD <> 0 THEN  
  BEGIN  
    ERROR_HANDLER(1);  
    EXIT;  
  END  
IF SETUP(NBUF,NKEYS,NNSEC,NDATF) <> 0 THEN  
  BEGIN  
    ERROR_HANDLER(2);  
    EXIT;  
  END;  
  
(* -----  
      Open Files  
-----*)  
*)  
FILE_NO := -1;      (* Automatic file number assignment *)  
RECORD_LEN := 32;  
FILE_NAME := 'K:PART.DAT';  
FILE_NO := OPNDAT(FILE_NO,S_FILE,FILEF_NAME,RECORD_LEN);  
  
IF ERRCOD <> 0 THEN  
  BEGIN  
    ERROR_HANDLER(3);  
    EXIT;  
  END  
IF LOKCOD <> 0 THEN  
  BEGIN  
    LOCK_CONFLICT(3);  
    EXIT;  
  END;  
  
(* -----  
      Initialize Data Buffer Pointer  
-----*)  
*)  
DATBUF_PTR := ADDR(DAT_BUFFER);
```

Listing 4-1. (continued)

```

(*) -----
      Set Exclusive Lock on Data Record No. 65686
-----
*)
DRN2 := 1;
SETDAT(DRN2);          (* Set two high-order bytes to 1,
                       which implies a base of 65536 *)
DRN := 150;           (* 65686 = 65536 + 150 *)
IF SETLOK(FILE_NO,X_LOCK,DRN) <> 0 THEN
  BEGIN
    LOCK_CONFLICT(4);
    EXIT;
  END;

(*) -----
      Read Data Record
-----
*)
SETDAT(DRN2);
IF READAT(FILE_NO,DRN,DATBUF_PTR) <> 0 THEN
  BEGIN
    ERROR_HANDLER(4);
    EXIT;
  END;

(*) -----
      Update Data Record
-----
*)
DAT_BUFFER.PART_QUAN := DAT_BUFFER.PART_QUAN - 100.00;

(*) -----
      Write Updated Record
-----
*)
SETDAT(DRN2);
IF WRDAT(FILE_NO,DRN,DATBUF_PTR) <> 0 THEN
  BEGIN
    ERROR_HANDLER(5);
    EXIT;
  END;

(*) -----
      Release Record Lock
-----
*)
SETDAT(DRN2);
IF FRELOK(FILE_NO,X_LOCK,DRN) <> 0 THEN
  BEGIN
    LOCK_CONFLICT(6);
    EXIT;
  END;

(*) -----
      Close Data File and Release File Lock
-----
*)
IF CLSDAT(FILE_NO) <> 0 THEN
  ERROR_HANDLER(7);
ELSE IF FRELOK(FILE_NO,S_FILE,0) <> 0 THEN
  LOCK_CONFLICT(7);

END.

```

Listing 4-1. (continued)

#### 4.6 Pascal/MT+ DATABASE Source Code

Your Access Manager distribution disk contains sample code for building and maintaining a data base in Pascal/MT+. The code is designed so you can add or substitute your own key attributes as required. The sample code is on your distribution disk in a file called DATABASE.SRC.

DATABASE.SRC demonstrates the integration of Access Manager with Pascal/MT+ applications. It builds a name and address data base and provides facilities for examining, updating, and/or listing the information contained therein. You might also want to use routines from DATABASE.SRC directly in your application programs.

[SINGLE] To create DATABASE.COM, compile DATABASE.SRC with MTPPLUS.COM and link as follows:

```
LINKMT DATABASE,AM80PASC/S,AM80BUF,PASLIB/S/D:8500
```

[MULTI] In a multiuser environment, your link statement should be entered as follows:

```
LINKMT DATABASE,AMQ8PASC,PASLIB/S
```

Note that Listing 4-2 of DATABASE.SRC might not include recent changes. You should always treat the copy on your distribution disk as the definitive version.

```

PROGRAM DATABASE;
(* .....
    DATABASE EXAMPLE VERSION 1.05 4/21/82 0751
    .....
*)
CONST
    MAX_KEY = 2;
    MAX_FIELD = 7;
    MAX_KEY_LEN = 20;
    MAX_FLD_LEN = 20;
    NAME_LEN = 14;
    FLD_NAME_LEN = 18;
    ACTION_LEN = 4;
    NEW_MODE = 1;
    OLD_MODE = 2;
    YES = 1;
    NO = 0;
    SAVE = 1;
    DELT = 2;
    BACK = 3;
    CONT = 4;
    STOP = 5;

TYPE
    KEYSTR = STRING[MAX_KEY_LEN];
    FLDSTR = STRING[MAX_FLD_LEN];
    FLDCHR = ARRAY[1..MAX_FLD_LEN] OF CHAR;

    CUST_REC = RECORD;
        CDF : CHAR;
        FLD : ARRAY[1..99] OF CHAR;
    END;

(*
    WORKING VARIABLES
*)
VAR
    KEY, TERMINAL, TRAP_ERRORS, TIME_OUT_TEST_DELAY, NO_BUFFERS : INTEGER;
    NO_NODE_SECTORS, NO_DATA_FILES, NO_KEYZ, FILE_NO : INTEGER;
    RECORD_LENGTH : INTEGER;
    SET_LENGTH, IDX_KEY, SPACE : KEYSTR;
    OLD_ACTION : INTEGER;
    FILNAME : STRING[NAME_LEN];
    NULL_BYT : BYTE;
    NULL_CHR : CHAR;

(*
    .....

```

Listing 4-2. DATABASE.SRC Source Code

Access Manager Programmer's Guide 4.6 Pascal/MT+ DATABASE Code

```

DATABASE FIELD & KEY DESCRIPTORS
*):
.....
*)
  DATBUF : CUST_REC;
  DATBUF_PTR : ^CUST_REC;
  FLD_NAME,KEY_NAME : ARRAY[0..MAX_FIELD] OF STRING[FLD_NAME_LEN];
  FLD_LEN : ARRAY[0..MAX_FIELD] OF BYTE;
  OLD_FLD,NEW_FLD : ARRAY[0..MAX_FIELD] OF FLDSTR;
  NO_FIELDS : INTEGER;

  IDX_NAME : ARRAY[0..MAX_KEY] OF STRING[NAME_LEN];
  KEY_LEN,KEY_MAP,KEY_TYPE,KEY_NUM,KEY_DUP : ARRAY[0..MAX_KEY] OF
    INTEGER;
  FOR_EVER : BOOLEAN;
  UNIQ_KEY,NLOCK,SLOCK,XLOCK,SFILE,XFILE,RLOCK : INTEGER;

(*
  INTERFACE TO AM80 (tm)

  AM80EXTR.PLI CONTAINS THE EXTERNAL DEFINITIONS OF THE AM-80 ROUTINES
*)

{$I AM80EXTR.PSC}

EXTERNAL FUNCTION @BDOS(FUNC:INTEGER; PARM:WORD) : INTEGER;

PROCEDURE GO_OP_SYS;

VAR
  DUMMY : INTEGER;
  DPARM : WORD;

  BEGIN
    DUMMY := @BDOS(0,DPARM);
  END;{GO_OP_SYS}

PROCEDURE DATA_BASE;

  BEGIN
    CLRSCR;

    CASE MAIN_MENU OF
1: DBNEW;
2: DBSCAN;
3: DBLIST;
4: DBSTAT;
5: DBSAVE;
6: DBTERM;
  END;{OF CASE}
  END;{DATA_BASE}

```

Listing 4-2. (continued)



```

(*)
:.....:
      ENTER NEW CUSTOMERS
:.....:
*)

PROCEDURE DBNEW;
VAR
  KEY,LOCK_CODE,NDRN,DRN,CHOICE : INTEGER;
  SAVE_KEY,LDRN,NO_LISTED : INTEGER;
  ROUTE : CHAR;
  CONTINUE,STAYPUT : BOOLEAN;
  L_VALUE,U_VALUE,CONV_TARGET,TARGET : KEYSTR;
  ACTION : INTEGER;

BEGIN
  ACTION := SAVE;
  WHILE (ACTION = SAVE) DO
    BEGIN
      ACTION := NEWDAT;
      LOCK_CODE := 0;
      IF ACTION = SAVE THEN
        BEGIN
          NDRN := UPDATE(0);
          LOCK_CODE := FRELOK(FILE_NO,XLOCK,NDRN);
          END;

          IF LOCK_CODE <> 0 THEN
            LOCK_TYPE(8);
          END;
        END;
      END;
    END;
  END;{DBNEW}

(*)
:.....:
      SCAN/UPDATE/DELETE CUSTOMERS
:.....:
*)

PROCEDURE DBSCAN;
VAR
  KEY,LOCK_CODE,NDRN,DRN,CHOICE : INTEGER;
  SAVE_KEY,LDRN,NO_LISTED : INTEGER;
  ROUTE : CHAR;
  CONTINUE,STAYPUT : BOOLEAN;
  L_VALUE,U_VALUE,CONV_TARGET,TARGET : KEYSTR;
  ACTION : INTEGER;

```

Listing 4-2. (continued)

```

BEGIN
KEY := SEARCH_KEY;
WRITELN;
WRITELN ('Enter target value for ',KEY_NAME[KEY],',');
WRITE (' or enter a period (.) to see main menu>>');
READLN(TARGET);
IF TARGET <> '.' THEN
BEGIN
CONV_TARGET := TARGET;
KEY_FORMAT(KEY,CONV_TARGET);
STAYPUT := TRUE;
WHILE (STAYPUT) DO
BEGIN
DRN := SERKEY(KEY_NUM[KEY],FILE_NO,SLOCK,
CONV_TARGET,IDX_KEY);
IF ERRCOD <> 0 THEN
ERROR_TYPE(KEY,2);
IF LOKCOD <> 0 THEN
STAYPUT := CHECK_LOCK(KEY,DRN)
ELSE
STAYPUT :=FALSE;
END;

OLD_ACTION := CONT;
CONTINUE := TRUE;
WHILE (CONTINUE) AND (DRN <> 0) DO
BEGIN
LDRN := DRN;
READ_CUST(DRN);
ACTION := OLDDAT(DRN);
SAVE_KEY := KEY;
IF ACTION = SAVE THEN
DRN := UPDATE(DRN);
IF ACTION = DELT THEN
DELETE(DRN);
IF (ACTION <> DELT) AND (FRELOK(FILE_NO,RLOCK,LDRN) <> 0)
THEN LOCK_TYPE(2);
IF (ACTION = SAVE) OR (ACTION = DELT) THEN
BEGIN
KEY := SAVE_KEY;
ACTION := OLD_ACTION;
END;
OLD_ACTION := ACTION;
CONV_TARGET := COPY(IDX_KEY,1,KEY_LEN[KEY]);
IDX_KEY := SET_LENGTH;
LOCK_CODE := 0;

STAYPUT := TRUE;
WHILE (STAYPUT) DO
BEGIN
IF ACTION = CONT THEN
BEGIN
DRN := AFTKEY(KEY_NUM[KEY],FILE_NO,
SLOCK, CONV_TARGET,IDX_KEY);
LOCK_CODE := LOKCOD;

```

**Listing 4-2. (continued)**

```

        END;
    IF ACTION = BACK THEN
    BEGIN
        DRN := BEFKEY(KEY_NUM[KEY],FILE_NO,
                    SLOCK, CONV_TARGET,IDX_KEY);
        LOCK_CODE := LOKCOD;
    END;

    IF LOCK_CODE <> 0 THEN
        STAYPUT := CHECK_LOCK(KEY,DRN)
    ELSE
        STAYPUT := FALSE;
    END;

    IF ACTION = STOP THEN
        CONTINUE := FALSE;
    END;

    WRITELN;
    WRITELN('SCAN ENDED');
    PAUSE;
    END;
END;{DBSCAN}

(*
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
                                LIST CUSTOMERS
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
*)

PROCEDURE DBLIST;
VAR
    KEY,LOCK_CODE,NDRN,DRN,CHOICE : INTEGER;
    SAVE_KEY,LDRN,NO_LISTED : INTEGER;
    ROUTE : CHAR;
    CONTINUE,STAYPUT : BOOLEAN;
    L_VALUE,U_VALUE,CONV_TARGET,TARGET : KEYSTR;
    ACTION : INTEGER;

BEGIN
    KEY := SEARCH_KEY;
    WRITELN;
    WRITE('Do you want listing routed to printer (Y/N) >>');
    READLN(ROUTE);
    IF ROUTE = 'y' THEN ROUTE := 'Y';

    WRITELN;
    WRITELN;
    WRITELN(
'Enter lower and upper limits for ',KEY_NAME[KEY],' listing;');
    WRITE('      separate values with a space >>') ;
    READ(L_VALUE);

```

**Listing 4-2. (continued)**



```

CLRSR;
WRITELN(FILENAME,' has ',GETDFS(FILE NO),
        ' records; currently, ',GETDFU(FILE_NO),
        ' of them are in use.');
```

```

WRITELN;
WRITELN;
WRITELN('      INDEX','ENTRIES':30);
WRITELN ('-----','-----':22);
FOR KEY := 0 TO MAX_KEY DO
    WRITELN(KEY_NAME[KEY]:16,' ':16,NOKEYS(KEY):7);

WRITELN;
WRITELN;
PAUSE;
END; {DBSTAT}

(*
::::::::::::::::::::::::::::::::::::::::::::::::::
                SAVE DATABASE UPDATES & RESTART
::::::::::::::::::::::::::::::::::::::::::::::::::
*)

PROCEDURE DBSAVE;
VAR
    KEY,LOCK_CODE,NDRN,DRN,CHOICE : INTEGER;
    SAVE_KEY,LDRN,NO_LISTED : INTEGER;
    ROUTE : CHAR;
    CONTINUE,STAYPUT : BOOLEAN;
    L_VALUE,U_VALUE,CONV_TARGET,TARGET : KEYSTR;
    ACTION : INTEGER;

BEGIN
    IF SAVDAT(FILE NO) <> 0 THEN
        ERROR_TYPE(0,7);
    FOR KEY := 0 TO MAX_KEY DO
        IF SAVIDX(KEY_NUM[KEY]) <> 0 THEN
            ERROR_TYPE(KEY,3);
    END; {DBSAVE}

(*
::::::::::::::::::::::::::::::::::::::::::::::::::
                SAVE DATABASE UPDATES & TERMINATE
::::::::::::::::::::::::::::::::::::::::::::::::::
*)

PROCEDURE DBTERM;
VAR
    KEY,LOCK_CODE,NDRN,DRN,CHOICE : INTEGER;
    SAVE_KEY,LDRN,NO_LISTED : INTEGER;
    ROUTE : CHAR;
```

Listing 4-2. (continued)

```

CONTINUE,STAYPUT : BOOLEAN;
L_VALUE,U_VALUE,CONV_TARGET,TARGET : KEYSTR;
ACTION : INTEGER;

BEGIN
  IF CLSDAT(FILE_NO) <> 0 THEN
    ERROR_TYPE(0,15);
  FOR KEY := 0 TO MAX_KEY DO
    IF CLSIDX(KEY_NUM[KEY]) <> 0 THEN
      ERROR_TYPE(KEY,16);
  IF FRELOK(FILE_NO,SFILE,0) <> 0 THEN
    ERROR_TYPE(0,13);

  WRITELN;
  WRITELN(' *** SUCCESSFUL TERMINATION ***');
  FOR EVER := FALSE;
END;{DBTERM}

(*
+++++
BEGINNING OF UTILITY FUNCTIONS

::::::::::::::::::::::::::::::::::::::::::::::::::
CLEAR SCREEN ROUTINE

::::::::::::::::::::::::::::::::::::::::::::::::::
*)

PROCEDURE CLRSCR;
VAR
  DUMMY : INTEGER;

  BEGIN
  FOR DUMMY := 1 TO 24 DO
    WRITELN;
  END;{CLRSCR}

(*
::::::::::::::::::::::::::::::::::::::::::::::::::
MAIN MENU ROUTINE

::::::::::::::::::::::::::::::::::::::::::::::::::
*)

FUNCTION MAIN_MENU : INTEGER;
VAR
  OP : INTEGER;

  BEGIN

```

Listing 4-2. (continued)

```

WRITELN(' :19,' AM-80(tm) DEMONSTRATION');
WRITELN;
WRITELN(' :19,' Customer Database Operations') ;
WRITELN(' :19,' Terminal ',TERMINAL);
WRITELN(' :19,' *****');
WRITELN;
WRITELN;
WRITELN(' 1. Enter New Customers');
WRITELN(' 2. Scan/Update/Delete Customer Records');
WRITELN(' 3. List Customer Records');
WRITELN(' 4. Database Statistics');
WRITELN(' 5. Save All Files & Restart Operations');
WRITELN(' 6. Terminate Operations');

OP := 0;
WHILE (OP < 1) OR (OP > 6) DO
  BEGIN
    WRITELN;
    WRITE('Enter desired operation number>>');
    READLN(OP);
  END;
MAIN_MENU := OP;
END; {MAIN_MENU}

(*)
:.....:
      SELECT SEARCH KEY ROUTINE
:.....:
*)
FUNCTION SEARCH_KEY : INTEGER;
VAR
  KEY,KEY_NO : INTEGER;

  BEGIN
    CLRSCR;
    WRITELN(' :24,'Customer Database Search Keys');
    WRITELN;
    WRITELN;
    WRITELN;

    FOR KEY := 0 TO MAX_KEY DO
      BEGIN
        KEY_NO := KEY + 1;
        WRITELN(KEY_NO,' - ',KEY_NAME[KEY])
      END;

    KEY := 0;
    WHILE (KEY < 1) OR (KEY > NO_KEYZ) DO
      BEGIN
        WRITELN;
        WRITELN;
        WRITE('Enter desired key number>>');

```

Listing 4-2. (continued)

```

        READLN(KEY);
        END;
    SEARCH_KEY := KEY-1;
    END; {SEARCH_KEY}

(*)
:.....

        ERROR HANDLING

:.....
*)

PROCEDURE ERROR_TYPE(INFO,LOCALE : INTEGER);
VAR
    DUMMY : INTEGER;

PROCEDURE ET_CLOSE;
VAR
    T_KEY : INTEGER;

    BEGIN
        DUMMY := CLSDAT(FILE NO);
        FOR T_KEY := 0 TO MAX_KEY DO
            IF T_KEY <> INFO THEN DUMMY := CLSIDX(KEY_NUM[T_KEY]);
        ET_STOP;
        END;

PROCEDURE ET_PCLOSE;
VAR
    L_KEY,T_KEY : INTEGER;

    BEGIN
        L_KEY := INFO + 1;
        IF L_KEY > MAX_KEY THEN GO_OP_SYS;
        FOR T_KEY := L_KEY TO MAX_KEY DO
            DUMMY := CLSIDX(KEY_NUM[T_KEY]);
        END;

PROCEDURE ET_STOP;

    BEGIN
        WRITELN;
        WRITELN('DATABASE TERMINATING WITH ERROR CODE #',ERRCOD);
        GO_OP_SYS;
        END;

    BEGIN
        WRITELN;
        WRITELN;
        WRITE('User Error #',ERRCOD,' occurred while trying to ');

        CASE LOCALE OF

```

Listing 4-2. (continued)



```

1:      WRITELN('open ',IDX_NAME[INFO]);
2:      WRITELN('search ',KEY_NAME[INFO],' Index File');
3:      WRITELN('save ',IDX_NAME[INFO]);
4:      WRITELN('remove old key from ',IDX_NAME[INFO]);
5:      WRITELN('enter key into ',IDX_NAME[INFO]);
6:      WRITELN('delete key from ',IDX_NAME[INFO]);
7: BEGIN
      WRITELN('save ',FILNAME);
      INFO := -1;
END;
8:      WRITELN('get a new data record',' (' ,FILE_NO,')');
9:      WRITELN('delete data record #',INFO);
10:     WRITELN('open ',FILNAME,' (' ,FILE_NO,')');
11:     WRITELN('read data record #',INFO);
12:     WRITELN('write data record #',INFO);
13:     WRITELN('release shared file lock on ',FILNAME);
14: BEGIN
      WRITELN('initialize user. ');
      GO_OP_SYS;
END;
15: BEGIN
      WRITELN('close ',FILNAME);
      INFO := -1;
END;
16:     WRITELN('close ',IDX_NAME[INFO]);

      END;{OF CASE}

      IF (LOCALE = 1) OR ((LOCALE > 7) AND (LOCALE < 14)) THEN
          ET_STOP
      ELSE IF (LOCALE = 2) OR ((LOCALE > 3) AND (LOCALE < 7)) THEN
          ET_CLOSE
      ELSE
          ET_PCLOSE;

      END;{ERROR_TYPE}

PROCEDURE LOCK_TYPE(LOCALE : INTEGER);
VAR
      T_KEY,DUMMY : INTEGER;

      BEGIN
      WRITELN('Lock Type: ',LOCALE,' Lock Code:',LOKCOD);
      DUMMY := CLSDAT(FILE_NO);
      FOR T_KEY := 0 TO MAX_KEY DO
          DUMMY := CLSIDX(KEY_NUM[T_KEY]);
      GO_OP_SYS;
      END;{LOCK_TYPE}

(*
:.....:
      READ DATA RECORD ROUTINE

```

**Listing 4-2. (continued)**

```

*)
:.....
PROCEDURE READ_CUST(DRN : INTEGER);
VAR
  FLD,CHR,POS_PTR : INTEGER;
  TMPBYT : ^BYTE;
  TMP_FLD : FLDSTR;
  FLDPTR : ^FLDCHR;

  BEGIN
  IF READAT(FILE_NO,DRN,DATBUF_PTR) <> 0 THEN
    ERROR_TYPE(DRN,11);

  TMPBYT := ADDR(TMP_FLD); (* POINTER LENGTH BYTE OF STRING *)
  FLDPTR := ADDR(TMP_FLD) + 1; (* PTR TO BODY OF STRING *)

  POS_PTR := 0;
  FOR FLD := 0 TO MAX_FIELD DO
    BEGIN
    TMPBYT^ := FLD_LEN[FLD];
    FOR CHR := 1 TO TMPBYT^ DO
      FLDPTR^ [CHR] := DATBUF.FLD[POS_PTR + CHR];

      WHILE (FLDPTR^[TMPBYT^] = ' ') AND (TMPBYT^ > 0) DO
        TMPBYT^ := TMPBYT^ - 1;
        OLD_FLD[FLD] := TMP_FLD;
        POS_PTR := POS_PTR + FLD_LEN[FLD];
      END;

    END; {READ_CUST}

  (*
  :.....
  LIST CUSTOMER RECORD ROUTINE
  :.....
  *)
PROCEDURE PRINT_CUST(ROUTE : CHAR);
VAR
  DUMMY : INTEGER;
  LIST_FILE : TEXT;

  BEGIN
  IF ROUTE = 'Y' THEN
    ASSIGN(LIST_FILE,'LST:');
  ELSE
    ASSIGN(LIST_FILE,'CON:');

  WRITELN(LIST_FILE);
  WRITELN(LIST_FILE,' ':4,OLD_FLD[0]:10,OLD_FLD[7]);
  WRITELN(LIST_FILE,' ':24,OLD_FLD[1],' ',OLD_FLD[2]);

```

Listing 4-2. (continued)

```

WRITELN(LIST_FILE,' ':24,OLD_FLD[3]);
WRITELN(LIST_FILE,' ':24,OLD_FLD[4],', ',OLD_FLD[5],', ',OLD_FLD[6]);
WRITELN(LIST_FILE);
CLOSE(LIST_FILE,DUMMY);
END;{PRINT_CUST}

(*
:.....
          PAUSE ROUTINE
:.....
*)
PROCEDURE PAUSE;
VAR
    NULL : CHAR;
    BEGIN
    WRITE('Press "RETURN" to continue ---');
    READLN(NULL);
    END;{PAUSE}

(*
:.....
          CONVERT TARGET VALUE TO KEY FORMAT ROUTINE
:.....
*)
PROCEDURE KEY_FORMAT(KEY : INTEGER;VAR TARGET : KEYSTR);
VAR
    TEMP : STRING[40];
    BEGIN
    IF UNIQ_KEY = KEY THEN
        EXIT
    ELSE
        BEGIN
        TEMP := CONCAT(TARGET,SPACE);
        TEMP :=COPY(TEMP,1,KEY_LEN[KEY]-2);
        TARGET := CONCAT(TEMP,NULL_CHR,NULL_CHR);
        END;
    END;{KEY_FORMAT}

(*
:.....
          COMPARE IDX_KEY & U_VALUE ROUTINE
:.....
*)
FUNCTION COMPARE(KEY : INTEGER; IDXVAL,UPVAL : KEYSTR) : INTEGER;

```

**Listing 4-2. (continued)**

```

VAR
    KL : INTEGER;
    C1,C2 : STRING[40];

    BEGIN
    IF KEY = UNIQ_KEY THEN
        KL := KEY_LEN[KEY]
    ELSE
        KL := KEY_LEN[KEY]-2;

    C1 := CONCAT (IDXVAL,SPACE);
    C1 := COPY (C1,1,KL);
    C2 := CONCAT (UPVAL,SPACE);
    C2 := COPY (C2,1,KL);

    IF C1<C2 THEN
        COMPARE := -1
    ELSE IF C1>C2 THEN
        COMPARE := 1
    ELSE
        COMPARE := 0;
    END; {COMPARE}

    (*
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        CHECK LOCK ROUTINES
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    *)

PROCEDURE SKIP_LOCK(KEY,DRN : INTEGER);
VAR
    L_VALUE : KEYSTR;

    BEGIN
    WHILE (DRN <> 0) AND (LOKCOD <> 0) DO
        BEGIN
        L_VALUE := COPY (IDX_KEY,1,KEY_LEN[KEY]);
        IDX_KEY := SET LENGTH;
        DRN := AFTKEY (KEY_NUM[KEY],FILE_NO,SLOCK,
            L_VALUE,IDX_KEY);
        END;
    END; {SKIP_LOCK}

FUNCTION CHECK_LOCK(KEY,DRN : INTEGER) : BOOLEAN;
VAR
    CONV_TARGET : KEYSTR;
    DUMMY : CHAR;

    BEGIN
    WRITELN;
    WRITE(
'Enter a "W" if you want to wait for locked record(s)>>');
    READLN (DUMMY);

```

Listing 4-2. (continued)

```

IF (DUMMY = 'W') OR (DUMMY = 'w') THEN
  BEGIN
    CHECK_LOCK := TRUE;
    EXIT;
    END;

WHILE (DRN <> 0) AND (LOKCOD <> 0) DO
  BEGIN
    CONV_TARGET := COPY (IDX_KEY,1,KEY_LEN[KEY]);
    IDX_KEY := SET_LENGTH;
    IF OLD_ACTION = CONT THEN
      DRN := AFTKEY (KEY_NUM[KEY],FILE_NO,
        SLOCK, CONV_TARGET,IDX_KEY)
    ELSE
      DRN := BEFKEY (KEY_NUM[KEY],FILE_NO,
        SLOCK, CONV_TARGET,IDX_KEY);
    END;
  CHECK_LOCK := FALSE;
  END;{CHECK_LOCK}

(*)
:.....:
      WARNING MESSAGES
:.....:
*)

PROCEDURE WARNING_TYPE (KEY,LOCALE,RET_CODE : INTEGER);
  BEGIN
    WRITELN;
    WRITE ('WARNING...Return Code #',RET_CODE,
      ' occurred while trying to ');

    CASE LOCALE OF
1:   WRITELN ('remove old key from ',IDX_NAME[KEY]);
2:   WRITELN ('enter key into ',IDX_NAME[KEY]);
3:   WRITELN ('delete key from ',IDX_NAME[KEY]);

    END; {OF CASE}
    PAUSE;
    END;{WARNING_TYPE}

(*)
:.....:
      ADD NEW KEY VALUE ROUTINE
:.....:
*)

```

Listing 4-2. (continued)

```

PROCEDURE ADD_A_KEY(KEY,DRN : INTEGER);
VAR
    RET_CODE,K_FLD : INTEGER;
    BEGIN
    K_FLD := KEY_MAP[KEY];
    (*
    REMOVE OLD KEY VALUE
    *)
    RET_CODE := DELKEY(KEY_NUM[KEY],FILE_NO,
        XLOCK,OLD_FLD[K_FLD],DRN);
    IF ERRCOD <> 0 THEN
        ERROR_TYPE(KEY,4);
    IF LOKCOD <> 0 THEN
        LOCK_TYPE(6);
    IF RET_CODE <> 1 THEN
        WARNING_TYPE(KEY,1,RET_CODE);
    (*
    ADD NEW KEY VALUE
    *)
    RET_CODE := ADDKEY(KEY_NUM[KEY],FILE_NO,
        XLOCK,NEW_FLD[K_FLD],DRN);
    IF ERRCOD <> 0 THEN
        ERROR_TYPE(KEY,5);
    IF LOKCOD <> 0 THEN
        LOCK_TYPE(7);
    IF RET_CODE <> 1 THEN
        WARNING_TYPE(KEY,2,RET_CODE);
    END; {ADD_A_KEY}
    (*
    ::::::::::::::::::::::::::::::::::::::::::::::::::::
        WRITE NEW DATA RECORD ROUTINE
    ::::::::::::::::::::::::::::::::::::::::::::::::::::
    *)
PROCEDURE WRITE_CUST(DRN : INTEGER);
VAR
    FLD,CHR,POS_PTR : INTEGER;
    TMPBYT : ^BYTE;
    TMP_FLD : FLDSTR;
    FLDPTR : ^FLDCHR;
    BEGIN
    DATBUF.CDF := NULL_CHR;

```

Listing 4-2. (continued)

```

TMPBYT := ADDR(TMP_FLD); (* POINTER LENGTH BYTE OF STRING *)
FLDPTR := ADDR(TMP_FLD) + 1; (* PTR TO BODY OF STRING *)

POS_PTR := 0;
FOR FLD := 0 TO MAX_FIELD DO
  BEGIN
    TMP_FLD := NEW_FLD[FLD];
    FOR CHR := 1 TO TMPBYT DO
      DATBUF.FLD[POS_PTR + CHR] := FLDPTR^ [CHR];

    WHILE (TMPBYT^ < FLD_LEN[FLD]) DO
      BEGIN
        TMPBYT^ := TMPBYT^ + 1;
        DATBUF.FLD[POS_PTR + TMPBYT^] := ' ';
      END;
    POS_PTR := POS_PTR + FLD_LEN[FLD];
  END;

IF WRDAT (FILE_NO,DRN,DATBUF_PTR) <> 0 THEN
  ERROR_TYPE (DRN,12);
END; {WRITE_CUST}

(*
.....
      DELETE KEY VALUE FROM INDEX ROUTINE
.....
*)
PROCEDURE DEL_A_KEY (KEY,DRN : INTEGER);
VAR
  RET_CODE,K_FLD : INTEGER;

  BEGIN
    K_FLD := KEY_MAP[KEY];

    RET_CODE := DELKEY (KEY_NUM[KEY],FILE_NO,
      XLOCK,OLD_FLD[K_FLD],DRN);

    IF ERRCOD <> 0 THEN
      ERROR_TYPE (KEY,6);
    IF LOKCOD <> 0 THEN
      LOCK_TYPE (10);
    IF RET_CODE <> 1 THEN
      WARNING_TYPE (KEY,3,RET_CODE);

  END; {DEL_A_KEY}

(*
.....
      UPDATE INDICES & DATA FILE ROUTINE
.....

```

Listing 4-2. (continued)

```

*)
FUNCTION UPDATE(DATA_RECORD : INTEGER) : INTEGER;
VAR
    FLD,KEY : INTEGER;

    BEGIN
    IF DATA_RECORD = 0 THEN
        BEGIN
        DATA_RECORD := NEWREC(FILE_NO,XLOCK);
        IF ERRCOD <> 0 THEN
            ERROR_TYPE(0,8);
        IF LOKCOD <> 0 THEN
            LOCK_TYPE(3) ;
        END;

        UPDATE := DATA_RECORD;
        FOR KEY := 0 TO MAX_KEY DO
            BEGIN
            FLD := KEY_MAP[KEY];
            IF OLD_FLD[FLD] <> NEW_FLD[FLD] THEN
                ADD_A_KEY(KEY,DATA_RECORD);
            END;

        FOR FLD := 0 TO MAX_FIELD DO
            IF OLD_FLD[FLD] <> NEW_FLD[FLD] THEN
                BEGIN
                WRITE_CUST(DATA_RECORD);
                EXIT;
                END;
            END;

        END;{UPDATE}

    (*
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        DELETE INDEX & DATA FILE ENTRY ROUTINE
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    *)
PROCEDURE DELETE(DATA_RECORD : INTEGER);
VAR
    FLD,KEY : INTEGER;

    BEGIN
    FOR KEY := 0 TO MAX_KEY DO
        BEGIN
        FLD := KEY_MAP[KEY];
        IF OLD_FLD[FLD] <> '' THEN
            DEL_A_KEY(KEY,DATA_RECORD);
        END;

    IF RETREC(FILE_NO,XLOCK,DATA_RECORD) <> 0 THEN
        ERROR_TYPE(DATA_RECORD,9);
    IF LOKCOD <> 0 THEN

```

Listing 4-2. (continued)



```

        LOCK_TYPE(9);
    END; {DELETE}

(*
    .....
        NEW DATA ENTRY ROUTINE
    .....
*)

FUNCTION NEWDAT : INTEGER;

VAR
    TMPFLD : STRING[40];
    UNIQUE : BOOLEAN;
    TMPDAT : INTEGER;
    FLD,OP_VAL,FLD_NO : INTEGER;
    OP1 : CHAR;
    OP1_BYT : BYTE;

LABEL
    l11;

BEGIN
    FOR FLD := 0 TO MAX_FIELD DO
        OLD_FLD[FLD] := '';

        CLRSCR;

        WRITELN(' :19,'Enter New Customer Information');
        WRITELN(' :19,'*****');
        WRITELN;
        WRITELN;
        WRITELN(
            ' [Press "RETURN" for customer # to see main menu.]');
        WRITELN;

        FOR FLD := 0 TO MAX_FIELD DO
            BEGIN
                FLD_NO := FLD + 1;

                WRITE(FLD_NO:6,' - ',FLD_NAME[FLD]:20,
                    '(' ,FLD_LEN[FLD]:2,') >>');
                READLN(NEW_FLD[FLD]);
                IF (FLD = KEY_MAP[UNIQ_KEY]) AND (NEW_FLD[FLD] = '')
                    THEN BEGIN
                        NEWDAT := STOP;
                        EXIT;
                        END;

                IF FLD = KEY_MAP[UNIQ_KEY] THEN
                    BEGIN
                        NEW_FLD[FLD] := CONCAT('0000',NEW_FLD[FLD]);
                    END;
            END;
        END;
    END;
l11:

```

Listing 4-2. (continued)

```

        RIGHT(NEW_FLD[FLD],FLD_LEN[FLD]);
        UNIQUE := TEST_UNIQUENESS;
        END
    ELSE
        BEGIN
            TMPFLD := CONCAT(NEW_FLD[FLD],
                ',');
            NEW_FLD[FLD] := COPY(TMPFLD,1,FLD_LEN[FLD]);
            UNIQUE := TRUE;
            END;

        IF ~UNIQUE THEN GOTO l11;
        END;

    WHILE (FOR EVER) DO
        BEGIN
            WRITELN;
            WRITELN;
            WRITELN;
            WRITELN(' ':19,'Current customer information');
            WRITELN;

            FOR FLD := 0 TO MAX_FIELD DO
                BEGIN
                    FLD_NO := FLD + 1;
                    WRITELN(FLD_NO:6,' - ',FLD_NAME[FLD]:20,' ',
                        NEW_FLD[FLD]);
                    END;

                OP_VAL := 0;
                WHILE (OP_VAL < 1) OR (OP_VAL > NO_FIELDS) DO
                    BEGIN
                        WRITELN;
                        WRITELN;
                        WRITELN(
                            'Enter S to save data, Field # to change data,');
                            WRITE(
                                'D to delete data, or E to end input >>');
                                READLN(OP1);
                                TMPDAT := 0;
                                IF (OP1 = 'S') OR (OP1 = 's') THEN
                                    TMPDAT := SAVE;
                                IF (OP1 = 'D') OR (OP1 = 'd') THEN
                                    TMPDAT := DELT;
                                IF (OP1 = 'E') OR (OP1 = 'e') THEN
                                    TMPDAT := STOP;
                                IF TMPDAT <> 0 THEN
                                    BEGIN
                                        NEWDAT := TMPDAT;
                                        EXIT;
                                        END;
                                    OP1_BYT := OP1;
                                    OP_VAL := OP1_BYT - 48;

```

Listing 4-2. (continued)

```

                END;
                UPDATE_FIELD(OP_VAL);
            END;

        END; {NEWDAT}

FUNCTION OLDDAT(DRN: INTEGER) : INTEGER;

VAR
    UNIQUE : BOOLEAN;
    TMPDAT : INTEGER;
    FLD,OP_VAL,FLD_NO : INTEGER;
    OPl : CHAR;
    OPl_BYT : BYTE;

BEGIN

    FOR FLD := 0 TO MAX_FIELD DO
        NEW_FLD[FLD] := OLD_FLD[FLD];

    CLRSCR;

    WHILE (FOR_EVER) DO
        BEGIN
            Writeln;
            Writeln;
            Writeln;
            Writeln(' :19,'Current customer information');
            Writeln;

            FOR FLD := 0 TO MAX_FIELD DO
                BEGIN
                    FLD_NO := FLD + 1;
                    Writeln(FLD_NO:6,' - ',FLD_NAME[FLD]:20,' ',
                        NEW_FLD[FLD]);
                END;

            OP_VAL := 0;
            WHILE (OP_VAL < 1) OR (OP_VAL > NO_FIELDS) DO
                BEGIN
                    Writeln;
                    Writeln;
                    Writeln(
'Enter C to continue scan, Field # to change data, S to save changes,');
                    WRITE(
'D to delete data, B for back scan, or E to end scan >>');
                    READLN(OPl);
                    TMPDAT := 0;
                    IF (OPl = 'C') OR (OPl = 'c') THEN
                        TMPDAT := CONT;
                    IF (OPl = 'S') OR (OPl = 's') THEN
                        TMPDAT := SET_XLOCK(OP1,DRN);
                    IF (OPl = 'D') OR (OPl = 'd') THEN
                        TMPDAT := SET_XLOCK(OP1,DRN);
                END;
            END;
        END;
    END;

```

Listing 4-2. (continued)

```

        IF (OP1 = 'B') OR (OP1 = 'b') THEN
            TMPDAT := BACK;
        IF (OP1 = 'E') OR (OP1 = 'e') THEN
            TMPDAT := STOP;
        IF TMPDAT <> 0 THEN
            BEGIN
                OLDDAT := TMPDAT;
                EXIT;
            END;
        OP1_BYT := OP1;
        OP_VAL := OP1_BYT - 48;
        END;
        UPDATE_FIELD(OP_VAL);

        END;
    END; {OLDDAT}

FUNCTION SET_XLOCK(OP : CHAR; DRN : INTEGER) : INTEGER;
VAR
    DUMMY : CHAR;

BEGIN
    DUMMY := 'W';
    WHILE (DUMMY = 'W') AND (SETLOK(FILE_NO,XLOCK,DRN) <> 0) DO
        BEGIN
            WRITELN;
            WRITELN('Customer update on hold due to record lock');
            WRITE(
'Enter W if you want to wait or any other key to cancel update>>');
            READLN(DUMMY);
            IF DUMMY = 'w' THEN DUMMY := 'W';
            END;

        IF DUMMY = 'W' THEN
            BEGIN
                IF OP = 'S' THEN
                    SET_XLOCK := SAVE
                ELSE
                    SET_XLOCK := DELT;
                END
            ELSE
                SET_XLOCK := OLD_ACTION;
            END; {SET_XLOCK}

(*
    :.....
        UPDATE DATA FIELD ROUTINE
    :.....
*)
PROCEDURE UPDATE_FIELD(FLD_NO : INTEGER);

```

**Listing 4-2. (continued)**

```

VAR
  TMPFLD : STRING[40];
  TEST : BOOLEAN;
  FIELD_NO : INTEGER;

BEGIN
  FIELD_NO := FLD_NO-1;
  TEST := FALSE;

  WHILE (~TEST) DO
    BEGIN
      WRITELN;
      WRITE('Input new ',FLD_NAME[FIELD_NO], '>>');
      READLN(NEW_FLD[FIELD_NO]);

      IF FIELD_NO = KEY_MAP[UNIQ_KEY] THEN
        BEGIN
          NEW_FLD[FIELD_NO] := CONCAT('0000',NEW_FLD[FIELD_NO]);
          RIGHT(NEW_FLD[FIELD_NO],FLD_LEN[FIELD_NO]);
        END
      ELSE
        BEGIN
          TMPFLD := CONCAT(NEW_FLD[FIELD_NO],
            ' ');
          NEW_FLD[FIELD_NO] := COPY(TMPFLD,1,FLD_LEN[FIELD_NO]);
        END;

      IF (FIELD_NO = KEY_MAP[UNIQ_KEY]) AND (NEW_FLD[FIELD_NO] <>
        OLD_FLD[FIELD_NO]) THEN
        TEST := TEST_UNIQUENESS
      ELSE
        TEST := TRUE;
      END;
    END; {UPDATE_FIELD}

  (*
  .....
  CUST # UNIQUENESS TEST ROUTINE
  .....
  *)

FUNCTION TEST_UNIQUENESS : BOOLEAN;
VAR
  TEMP : INTEGER;
  TEST : FLDSTR;

BEGIN
  TEST := NEW_FLD[KEY_MAP[UNIQ_KEY]];
  TEMP := GETKEY(UNIQ_KEY,0,NLOCK,TEST);

  IF LOKCOD <> 0 THEN
    LOCK_TYPE(12);

```

Listing 4-2. (continued)



```

FLD_LEN[5] := 2;
FLD_NAME[6] := 'Zipcode';
FLD_LEN[6] := 9;
FLD_NAME[7] := 'Customer Status';
FLD_LEN[7] := 8;

KEY_LEN[0]:=10;
KEY_TYPE[0]:=0;
KEY_MAP[0]:=2 ; (* KEY 0 = LAST NAME *)
KEY_LEN[1]:=11;
KEY_TYPE[1]:=0;
KEY_MAP[1]:=6 ; (* KEY 1 = ZIPCODE *)
KEY_LEN[2]:=4 ;
KEY_TYPE[2]:=0;
KEY_MAP[2]:=0 ; (* KEY 2 = CUST NUMBER *)

UNIQ_KEY := 2 ; (* USED IN TEST OF UNIQUENESS *)

FOR KEY := 0 TO MAX_KEY DO
BEGIN
  IF KEY = UNIQ_KEY THEN
    KEY_DUP[KEY] := NO
  ELSE
    KEY_DUP[KEY] := YES;

  KEY_NAME[KEY] := FLD_NAME[KEY_MAP[KEY]];
END;

IDX_NAME[0] := 'NAME.IDX';
IDX_NAME[1] := 'ZIPC.IDX';
IDX_NAME[2] := 'NUMB.IDX';

NLOCK := 0; (* IGNORE LOCKS *)
SLOCK := 1; (* SHARED RECORD LOCK *)
XLOCK := 2; (* EXCLUSIVE RECORD LOCK *)
SFILE := 3; (* SHARED FILE LOCK *)
XFILE := 4; (* EXCLUSIVE FILE LOCK *)
RLOCK := 5; (* RELEASE SLOCK) OR (XLOCK *)

(*
::::::::::::::::::::::::::::::::::::::::::::::::::
INITIALIZE INDEX FILES
::::::::::::::::::::::::::::::::::::::::::::::::::
*)

SET_LENGTH := '12345678901';
IDX_KEY := SET_LENGTH;
SPACE := ' ';
```

Listing 4-2. (continued)

```

*) SET TERMINAL TO -1 FOR AUTOMATIC ASSIGNMENT BY AM-80

TERMINAL := -1;
TRAP_ERRORS := YES;
TIME_OUT_TEST_DELAY := 2; (* APPROXIMATELY 2 SECONDS *)
TERMINAL := INTUSR(TERMINAL,TRAP_ERRORS,TIME_OUT_TEST_DELAY);
IF ERRCOD <> 0 THEN
    ERROR_TYPE(0,14);

NO_BUFFERS := 5;
NO_NODE_SECTORS := 4;
NO_DATA_FILES := 1;
NO_KEYZ := MAX_KEY + 1;

IF SETUP(NO_BUFFERS,NO_KEYZ,NO_NODE_SECTORS,NO_DATA_FILES) <> 0 THEN
    BEGIN
        WRITELN('Illegal SETUP Parameters');
        EXIT;
        END;

FOR KEY := 0 TO MAX_KEY DO
    BEGIN
        KEY_NUM[KEY] := OPNIDX(-1,IDX_NAME[KEY],
            KEY_LEN[KEY], KEY_TYPE[KEY],KEY_DUP[KEY]);
        IF ERRCOD <> 0 THEN
            ERROR_TYPE(KEY,1);
        END;

(*)
:.....:
INITIALIZE DATA FILE
:.....:
*)
FILE NO := -1;
RECORD_LENGTH := 100;
FILENAME := 'CUSTOMER.DAT';
FILE NO := OPNDAT(FILE_NO,SFILE,FILNAME,RECORD_LENGTH);
IF ERRCOD <> 0 THEN
    ERROR_TYPE(0,10);
IF LOKCOD <> 0 THEN
    LOCK_TYPE(1);

(*)
CUST_REC IS THE DATA FILE BUFFER AREA
*)
DATBUF_PTR := ADDR(DATBUF);

(*)
:.....:

```

Listing 4-2. (continued)



```
BEGIN DATABASE OPERATION
*):.....
NULL_BYT := 0;
NULL_CHR := NULL_BYT;
FOR_EVER := TRUE;
WHILE (FOR_EVER) DO
    DATA_BASE;
EXIT;
END.
```

**Listing 4-2. (continued)**

End of Section 4

# Index

## B

background server  
  CB80, 2-1  
  creating, 1-7  
  customization routines, 1-2  
  Pascal/MT+, 4-2  
  PL/I, 3-1  
  routines, 1-2  
buffer areas, 1-9

## C

code requirements, 1-3

## D

data file  
  CB80 example, 2-4  
  CB80 routines, 2-24  
  multiuser, 1-8  
  Pascal/MT+ example, 4-5  
  PL/I example, 3-5  
  status information, 1-4  
data record numbers, 1-4  
data record  
  record length, 1-4  
DATABASE sample  
  CB80 source code, 2-7  
  Pascal/MT+ source code, 4-9  
  PL/I source code, 3-9  
design constraints, 1-5  
distribution disk  
  contents, 1-1

## E

error messages, 1-7  
external declarations  
  components, 1-1  
  CB80, 2-2  
  Pascal/MT+, 4-2  
  PL/I, 3-2

## I

index file  
  multiuser, 1-8  
  record length, 1-4, 1-8

## K

key values  
  CB80 numeric values, 2-2  
  length, 1-4  
  minimum number of, 1-4  
  Pascal/MT+ numeric values, 4-2  
  PL/I numeric values, 3-3

## L

linking  
  CB80 programs, 2-1  
  Pascal/MT+ programs, 4-1  
  PL/I programs, 3-1

## M

multiuser module, 1-5

## P

parameters  
  PL/I, 3-2  
pointers, 1-4

## Q

queues  
  creating, 1-5  
  space requirements, 1-6

## R

RECREATE utility  
  CB80, 2-2  
  Pascal/MT+, 4-3  
  PL/I, 3-4  
RMAC assembler, 1-9

## S

shared routines  
  cancelling, 1-7