



M

VA M Programming Intermediate

Lesson 6

Enhanced Global File Structure and Techniques

Veterans Health Administration
Office of Information
OI National Training and Education Office

VA M PROGRAMMING

Intermediate

Table of Contents

Guide to Symbols Used	1
Optional Reading References	1
References	1
Document Update Note	1
Objectives	2
Enhancements to Our Mailing List System	2
"Generic" Menu	2
Routine names	3
M Code Sample	3
M Code Sample	3
M Code Sample	4
Number of Menu Items	4
M Code Sample	4
M Code Sample	4
Adding a Menu Option	4
Using the Generic Menu for Other Applications	5
Pseudocode for VECS6MN, the Menu Routine	5
M Code for VECS6MN, the Menu Routine	6
M Code Sample	6
New File Structure	7
File Header Node	7
Data Nodes	8
Creating the File	8
M Code Sample	9
The LOCK Command	9
M Code Sample	10
M Code Sample	10
M Code Sample	10
M Code Sample	11
M Code Sample	11
M Code Sample	11
"Ringin" the Bell!	11
M Code Sample	12
Special Form of the AND Operator	12
Lesson 5 M Code Sample:	12

VA M PROGRAMMING

Intermediate

M Code Sample	13
M Code Sample	13
Handling the Date	13
M Sample Code	14
Pseudocode for VECS6CR, the Create Routine	15
M Code for VECS6CR, the Create Routine	17
Editing the File	19
Finding an Individual	19
M Code Sample	19
Date Conversion	19
Pseudocode for VECS6ED, the Edit Routine	21
M Code for VECS6ED, the Edit Routine	24
M Code Sample	24
Deleting Records	26
M Code Sample	26
Devices	27
The OPEN Command	27
M Code Sample	27
The USE Command	28
M Code Sample	28
The CLOSE Command	28
M Code Sample	28
The \$IO Special Variable	28
Device Control in the VA	28
Pseudocode for VECS6PR, the Print Routine	29
M Code for VECS6PR, the Print Routine.	31
Limitations	32
Assignment (Optional)	33
In General	33
Important	33
Assignment Specifications	33

VA M PROGRAMMING

Intermediate

Acknowledgements

Developing a comprehensive training package for the **M** computer programming language as it is used in VA was not an easy task. Much gratitude goes to the members of the M Programming Language Training Development Team who spent countless hours developing this training program. Sincere appreciation to:

Debbie Channell
Tuscaloosa OI Field Office, Customer Services

Harris H. Lloyd
Central Arkansas Veterans Health Care System

Bob Lushene
Hines OI Field Office, Technical Services

VA M PROGRAMMING

Intermediate

Guide to Symbols Used

<ENTER> Press the Enter or Return key

<TAB> Press the Tab key

<space> One space



What follows is from part of the VA Programming Standards and Conventions.



This indicates an OPTIONAL reading reference. Additional information on this topic may be found in the references listed below.

Optional Reading References

Lewkowicz, John. *The Complete M: Introduction and Reference Manual for the M Programming Language*. 1989.

Walters, Richard. *M Programming – A Comprehensive Guide*. 1997.

The optional reading references are available through:

M Technology Association
1738 Elton Road, Suite 205
Silver Spring, MD 20903
301-431-4070

References

The VA M Programming Standards and Conventions shown were taken from “VA DHCP Programming Standards and Conventions” document, prepared March 19, 1991, modified in 1995 and approved January 22, 1996.

<http://vaww.vista.med.va.gov/Policies/sacc.htm>.

Document Update Note

January 2007 – To meet current standards on personal data privacy and accessibility, this document was updated. It was edited to include document metadata and edited to de-identify any personal data. Only the information in this document is affected. This should have no effect on the M routines.

Objectives

The objective of this lesson is to prepare you to put a complete system together using the following commands and techniques:

- Files structures using internal record numbers
- "Generic" menu that can be used for different applications and is easy to modify
- Locking files to preserve data integrity
- Device handling: switching between the screen and a printer
- "Ringing" the bell to draw the user's attention to a message
- A special form of the AND operator
- Recognizing the limitations of this version of the system

Enhancements to Our Mailing List System

As we mentioned in Lesson 5, the Mailing List System will be used as the basis for our efforts in this lesson as well as in the lessons to follow. In this lesson, we have enhanced the system in several ways:

1. We have changed the file structure so that duplicate names can be added. In addition, the new file structure will be closer to the file structure used in the VA.
2. We've made the menu more "generic" so that we can use it for any application and/or can add menu items by adding one line only.
3. Since we are using global files, which can be accessed by multiple users, we will lock the file to prevent someone from locking the same node at the same time.
4. We have included code to handle switching your input/output between a terminal screen and a printer.

"Generic" Menu

Most interactive applications start by presenting the user with a menu. Based on the user's choice, the system then goes to a particular function. When that function is completed, the system returns to the menu where the user can choose to exit or choose another option.

VA M PROGRAMMING

Intermediate

This procedure varies little from application to application. What does vary is the heading for the menu, the text for each option, and the name of the routines called for each function. Since that is the case, we are going to develop a menu driver routine that can be used by most any application with little modification. In addition, we want to be able to add or delete menu options without major modification to the menu routine.

Routine names

The basis for this menu driver is the OPTIONS paragraph. In addition to containing the text for the option, we have now added the name of the routine that is to be called when the user chooses that option number.

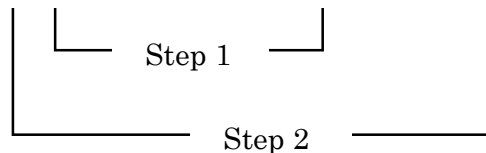
M Code Sample

```
OPTIONS ;
    ;;.....CREATE MAILING LIST ENTRY^^VECS6CR
    ;;.....EDIT MAILING LIST ENTRY^^VECS6ED
    ;;.....DELETE MAILING LIST ENTRY^^VECS6DEL
    ;;.....PRINT MAILING LIST ENTRIES^^VECS6PR1
```

Since these routines are external routines, they have an ^ in front of them. We have also used an ^ as the delimiter between the menu text and the routine name. Therefore, we need the following line of code to pull out the routine name with the necessary ^ in front of it:

M Code Sample

```
S ROUTINE=$P($T(OPTIONS+OPT), "^", 2, 3)
```



Step 1: M pulls out the line in the OPTIONS paragraph that relates to the OPT number entered by the user

Step 2: Piece 1 is the menu text. In order to pull out the routine name with the ^ in front of it, we must specify pieces 2 (empty) through 3 (the delimiter ^ is included followed by piece 3 (the routine name)) which is SET into ROUTINE.

VA M PROGRAMMING

Intermediate

Once the routine name is in the variable ROUTINE, we can use indirection to call it as shown in the M code sample below:

M Code Sample

```
D @ROUTINE G MAIN
```

Number of Menu Items

Another way in which we are going to make this menu generic is to remove any references to a fixed number of menu items. Instead, we'll let M keep track of how many menu items there are. Where we display the menu, we will now QUIT the FOR loop when the variable XDESCR (the menu text) is null which signals the end of the list. When the loop has been completed, we will take the loop counter (II), subtract 1 from it (the loop went one beyond the last actual menu item), and place the contents in the variable ITEMS:

M Code Sample

```
F II=1:1 D DISMENU2 Q:XDESCR=" "  
S ITEMS=II-1
```

Once we have the variable ITEMS, we can use it to check the range of valid item numbers:

M Code Sample

```
I (OPT<1)!(OPT>ITEMS) W !!,"OPTION NUMBER MUST BE "  
I W "BETWEEN 1 AND ",ITEMS G DISMENU1
```

Adding a Menu Option

With the routine names in the OPTIONS paragraph and the FOR loop providing a count of the number of items, the only thing we have to do to add an option to the menu is to insert a line in the OPTIONS paragraph in the position relative to the other options where you want the new option to appear on the menu. That OPTIONS line will, of course, contain the text for the option and the routine name to be called when the user selects that option.

VA M PROGRAMMING

Intermediate

Using the Generic Menu for Other Applications

We can use the same menu driver routine with another application by:

1. Changing the heading of the menu
2. Putting in the appropriate options in the OPTIONS paragraph

Pseudocode for VECS6MN, the Menu Routine

Here is a table containing a detailed explanation of the operation of the menu program using pseudocode. Review the table thoroughly.

<i>Line Labels</i>	<i>Steps</i>
MAIN+1	1. Display the menu and get the user's selection using the Display Menu subroutine (Step 7).
MAIN+2	2. If the user pressed the enter or ^ key, go to the Exit subroutine (Step 6).
MAIN+3	3. Using the option number that the user entered, get the routine name from the OPTIONS paragraph.
MAIN+4	4. Do the subroutine that corresponds to the user's option number.
MAIN+4	5. Go back to Step 1.
EXIT+1,+2	6. The Exit subroutine: KILL any variables and QUIT.
DISMENU	7. The Display Menu subroutine:
DISMENU+1	a. Clear the screen.
DISMENU+2	b. Write the menu heading.
DISMENU+4	c. Start a loop, which does the Display Menu 2 subroutine (Step 8). Stop the loop when there are no more options to display.
DISMENU+5	d. Set the number of items displayed equal to the number of iterations in the loop minus 1.
DISMENU1+1	e. Prompt the user to enter an option number.
DISMENU1+2	f. If the user times out, enters an ^, or presses the Enter, go to the Exit subroutine (Step 6).
DISMENU1+3:+4	g. If the option number the user entered is not one of the ones displayed, display an error message and go back to Step 7e.
DISMENU1+5	h. QUIT the subroutine.
DISMENU2	8. The Display Menu 2 subroutine:
DISMENU2+1	a. Set a variable with the text for the option taken from the OPTIONS paragraph.
DISMENU2+2	b. If the option text is not null, write the option number with the text for that option.
DISMENU2+3	c. QUIT the subroutine.

VA M PROGRAMMING

Intermediate

M Code for VECS6MN, the Menu Routine

The M code below is for the routine discussed in the preceding pages. Go back to the previous pages and review the pseudocode if the reason for any of the M code here is unclear.

M Code Sample

```
VECS6MN ;ATL/ACE PROGRAMMER-MAILING LIST SYSTEM ;12/1/90
; ;1
; VARIABLE LIST
;
; SPECIAL VARIABLES USED IN VA (not to be KILLed):
;     DTIME.....Delay time
;     IOF.....contains the codes to start output
;               at the top of a page or screen
; APPLICATION VARIABLES:
;     OPT.....Option number chosen by the user
;     ROUTINE...Routine name for option chosen
;     ITEMS.....Number of items in menu
;     II.....Menu loop counter
;     XDESCR....Text for menu option
;
MAIN ;
D DISMENU
I (OPT["^") G EXIT
S ROUTINE=$P($T(OPTIONS+OPT),"^",2,3)
D @ROUTINE G MAIN
EXIT ;
K OPT,ROUTINE,ITEMS,II,XDESCR
Q
DISMENU;
W @IOF
W !!!,?20,"MAILING LIST SYSTEM"
W !!,"OPTIONS     FUNCTIONS"
F II=1:1 D DISMENU2 Q:XDESCR=""
S ITEMS=II-1
DISMENU1 ;
R !!,"ENTER OPTION NUMBER: ",OPT:DTIME
IF '$T!(OPT["^")!(OPT="") S OPT="^" Q
I (OPT<1)!(OPT>ITEMS) W !!,"OPTION NUMBER MUST BE "
I W "BETWEEN 1 AND ",ITEMS G DISMENU1
Q
DISMENU2 ;
S XDESCR=$P($T(OPTIONS+II),";",2)
I XDESCR'="" W !!,II,$P(XDESCR,"^",1)
Q
```

VA M PROGRAMMING

Intermediate

```
OPTIONS ;
;;.....CREATE MAILING LIST ENTRY^^VECS6CR
;;.....EDIT MAILING LIST ENTRY^^VECS6ED
;;.....DELETE MAILING LIST ENTRY^^VECS6DEL
;;.....PRINT MAILING LIST ENTRIES^^VECS6PR
```

New File Structure

Because we used the individual's name as the subscript to our file in Lesson 5, we saw that two people with the same name could not be added to the file (the second entry would overwrite the first). In this lesson, we will assign record numbers to each individual and use that record number as the subscript, thus allowing two individuals with the same name to be stored with two different record numbers.

The exact file layout we will use is very similar to the structure used within the VA. In future lessons, we will develop the similarities even further.

File Header Node

In Lesson 4, we used an array element to hold the total number of entries that were made to the array, for example ARRAY(0). In this lesson, we are going to expand the use of this concept for global files and create what we will call the 0 node. The 0 node will contain four pieces:

^VECS6G(0)=file name^file number^last record nr.^nr. of entries

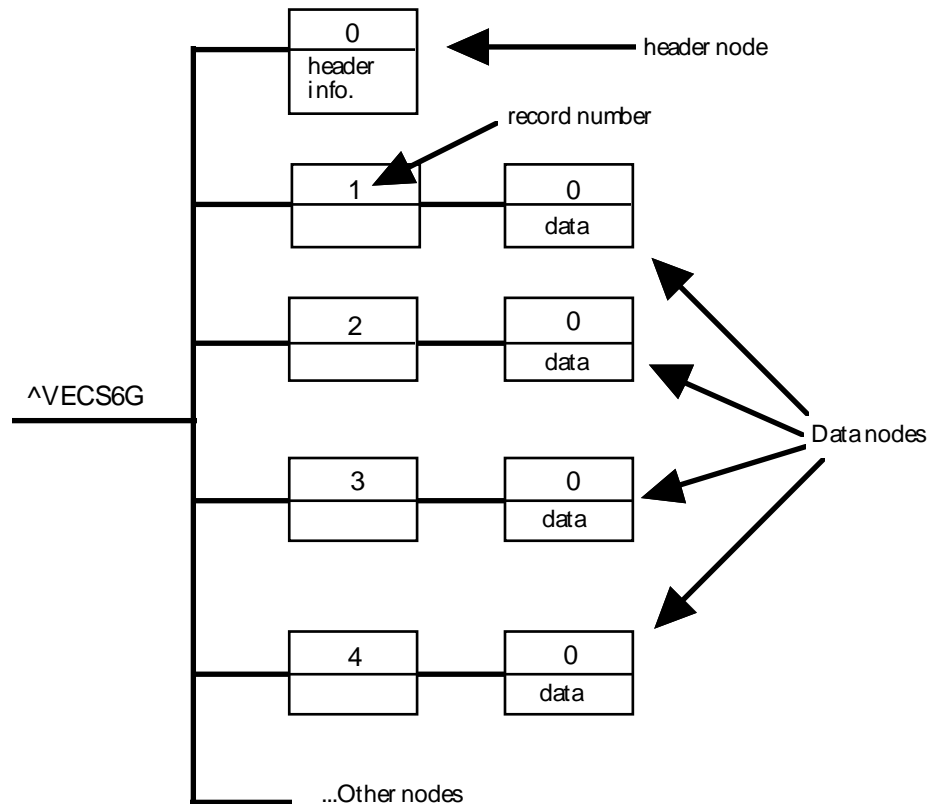
file name:	a short, descriptive name for the file
file number:	at this point, we will just use a consecutive number for each file
last record nr:	we will be assigning record numbers to each individual as they are entered; as we make the assignment, we must keep track of the last number used
nr. of entries:	the total number of entries made to the file

VA M PROGRAMMING

Intermediate

Data Nodes

The data will be stored in what we call second level nodes. When we specify a file reference with one subscript, we call that a first level node. We can have more than one subscript, however; each subscript represents an additional level on the tree diagram:



`^VECS6G(record number,0)=name^address^city^state^zip^phone^last contact date`

Creating the File

M code that could be used to add a record to a file is shown in the ADD paragraphs below. There are two basic differences between the routine to create this file and the routine to create the file in Lesson 5.

1. The most obvious is that the statement to set the data into the node will be different:

```
S ^VECS6G(RECNR,0)=NAME_"^"_ADDR_"^"_CITY_"^"_STATE_"^"_ZIP_"^"_PHONE_"^"_LCD
```

VA M PROGRAMMING

Intermediate

2. Since we have the header node now, we will have to make sure it is updated as necessary. Steps a through e below explain the ADD paragraph of the Create routine. The ADD paragraph is shown at the bottom of the display.

ADD+1

a. We need to lock the header node so that another user cannot be assigned the same record number if they are creating a record at the same time.

ADD+2

b. If we go to add a record and the header record does not yet exist (i.e., this is the first record of the file), we will create it.

ADD+3

c. We need to get a record number for the new entry. We look at the header record for the last record number assigned, we add 1 to it, and see if it is being used.

ADD+4

d. We need to update last record number and the number of entries.

ADD+5:+6

e. We create the data node with the individual's name and display the record number on the screen.

M Code Sample

```
ADD ;
L ^VECS6G(0):5 I '$T W !,"File is busy, try again later" S OK=0 Q
I '$D(^VECS6G(0)) S ^VECS6G(0)="MAILING LIST^1^0^0"
F RECNR=$P(^VECS6G(0),"^",3)+1:1 I '$D(^VECS6G(RECNR)) Q
S $P(^VECS6G(0),"^",3,4)=RECNR_"^"_$P(^VECS6G(0),"^",4)+1)
S ^VECS6G(RECNR,0)=NAME L
W !,"Record # ",RECNR,!
Q
```

The LOCK Command

The purpose of the LOCK command is to prevent multiple users from **simultaneously** LOCKing a node of a global file. Global files are usually stored on disk and available to multiple users at the same time. It is possible for one user to access a node while another user is updating the node.

VA M PROGRAMMING

Intermediate

When you LOCK a global node, you are locking all nodes below it as well (if there are any). When you have a node or branch LOCKed, no one else can LOCK the same node or branch. Notice that we said that no one else can LOCK the same node or branch. There is a potential problem here: a LOCKed node can be updated by another application if that routine does not use a LOCK! In another words, a LOCK is only effective if **every programmer in every routine** uses a LOCK when updating global files.

There are several forms of the LOCK command:

M Code Sample

```
LOCK ^VECS6G(0)
```

This form specifies a variable name or a reference to a node.
The reference to node ^VECS6G(0) will be entered into the LOCK table.

One caution: when M encounters this lock, it will first unLOCK all existing LOCKs for this routine. Then, if any variables listed are LOCKed by another process, M will pause until the variable is released.

M Code Sample

```
LOCK
```

This is called the **argumentless LOCK**. It will unLOCK all variables previously LOCKed by your job. If you do not specify an argumentless LOCK, one will be issued when you HALT.

M Code Sample

```
LOCK ^VECS6G(0):5  
IF '$T W !,"Record being updated...try again later" G EXIT
```

This form adds a timeout of five seconds (:05) to the node. If the LOCK command is successful within the time specified, \$TEST is set to 1; if the LOCK is not successful (the time runs out), \$TEST will be set to 0. If you specify multiple variable names or nodes on the LOCK and the LOCK is not successful, none of the variables or nodes will be LOCKed.

This form of the LOCK is highly recommended.

VA M PROGRAMMING

Intermediate

M Code Sample

```
LOCK +^VECS6G(0):5
```

```
IF '$T W !,"Record being updated...try again later" G EXIT
```

This form is called the incremental LOCK. The plus sign indicates that we want to LOCK one node only ^VECS6G(0). No nodes are unLOCKed.

M Code Sample

```
LOCK +^VECS6G(0),+^PAT(34,0)
```

Using the incremental LOCK avoids the problem we saw above with listing multiple node names on the LOCK. When we put the plus sign in front of several nodes, we will be able to LOCK each one--nodes are NOT unLOCKed first.

This form of the LOCK is highly recommended.

M Code Sample

```
LOCK -^VECS6G(0)
```

This form is called the decremental LOCK. It will unLOCK the ^VECS6G(0) node only.

This form of the LOCK is highly recommended.

"Ring" the Bell!

One technique that is often used to draw the user's attention to an error message is to "ring" a bell when the message is issued.

Using the WRITE command with an *7 is no longer the appropriate nor acceptable M code syntax to do this. 7 is the ASCII decimal code for the bell and to write the ASCII value, we must put the * in front of the decimal number.



VA Programming Standards and Conventions

Require that M code be portable to all platforms. The use of *7 (star 7) is not always interpreted correctly on all systems. Therefore, the correct syntax to be used should be WRITE \$C(7) or WRITE \$C(65), as shown in the M code samples.

VA M PROGRAMMING

Intermediate

M Code Sample

Inappropriate

```
WRITE *7
      Rings the bell

WRITE *65
      Writes the letter A
```

Appropriate

```
WRITE $C(7)
      Rings the bell

WRITE $C(65)
      Writes the letter A
```

Special Form of the AND Operator

In previous lessons we used the ampersand (&) as the AND operator to indicate that we want to combine truth values from two comparisons. We know that M would find the truth value for the first comparison, and then find the truth value for the second comparison. With those two truth values, M would use a table to get the resulting truth value:

```
1&1=1
1&0=0
0&1=0
0&0=0
```

To summarize, there is only one way to get a true (1) from the AND operation; that is from two true (1) values.

There are some times, however, that you would like M to check the first condition and only check the second condition if the first is true. For instance, in the create routine in this lesson you will be checking to see if the user pressed the ENTER key in response to a prompt for a field. If they do, then you do not want to go through any of the field's validation with the value null. In the last lesson, we avoided that by code as shown below:

Lesson 5 M Code Sample:

```
ADDR ;
  R !,"ADDRESS: ",ADDR:DTIME I '$T!(ADDR["^") G CREXIT
  I ADDR="" G CITY
  I ADDR'?..ANP!($L(ADDR)>25)!(ADDR["?")
  I W !,"Address must not exceed 25 characters",! G ADDR
CITY ;
```

Do you remember your pattern matches? This is really a case where a null would pass the validation logic since null does match dot ANP. Usually, a null does not pass the validation logic for a field.

VA M PROGRAMMING

Intermediate

If, for some reason, you were to leave out line ADDR+2 from the Lesson 5 M Code Sample on the previous page, you might try the code below:

M Code Sample

```
ADDR ;
R !, "ADDRESS: ", ADDR:DTIME I '$T!(ADDR["^") G CREXIT
I ADDR'=" "&(ADDR'?.ANP)!($L(ADDR)>25)!(ADDR["?")
I W !, "Address must not exceed 25 characters", ! G ADDR
```

What will happen if the variable ADDR is null? The truth value for the first comparison of ADDR+2 will be evaluated to a 0. We know that there is no way we will end up with a truth value of 1 for the combination of the remaining comparisons because with an AND operator, if the first comparison is false, there is no way to get a resulting true. Also, if ADDR is null, there's no reason to run through the validations--there's nothing to validate!

To avoid this problem, we use an alternative AND operator: the comma. When we use a comma instead of the ampersand (&), M will check the first comparison: if it is false (0), M will not check the second comparison, which is more efficient.

M Code Sample

```
ADDR ;
R !, "ADDRESS: ", ADDR:DTIME I '$T!(ADDR["^") G EDEXIT
I ADDR'=" ", ADDR'?.ANP!($L(ADDR)>25)!(ADDR["?")
I W !, "Address must not exceed 25 characters.", !, $(7) G ADDR
```

There is one restriction to the use of the comma instead of the ampersand (&): the comma can **only** be used on the IF command. That is, the comma **cannot** be used with postconditionals.

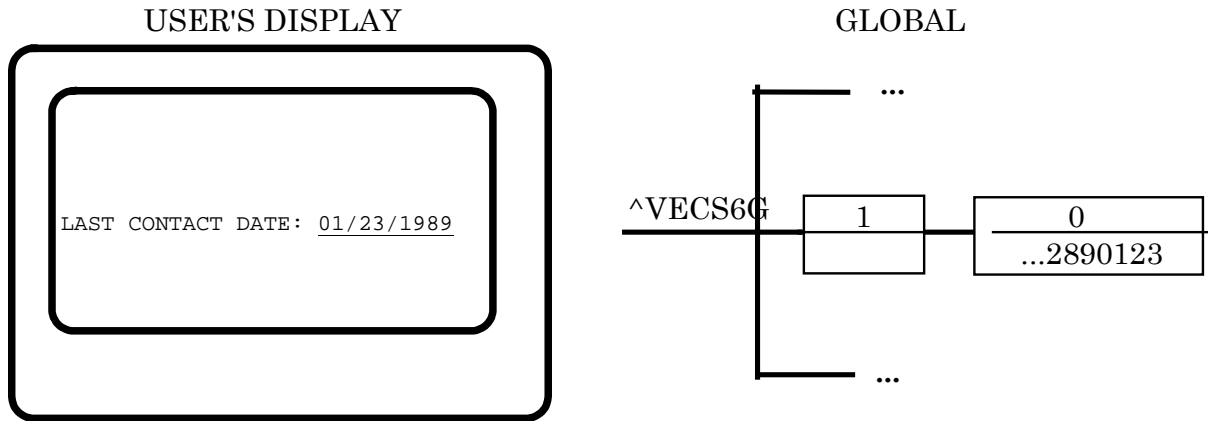
Handling the Date

It is common in computer software to store the data from some fields in a different format than is displayed to the user. One good example of this concept is the date. The user is accustomed to entering the date as MM/DD/YYYY where MM is the month, DD is the day, and YYYY is the year. We will call this the display format. However, if we made the date an M subscript so dates would be sorted, they would not come out in an order that would be useful to us. Instead, we will accept the user's input in display format and convert it to another format for storage called internal format: YYYYMMDD where YYYY is a designation for the year, MM is the month and DD is the day.

VA M PROGRAMMING

Intermediate

You will notice in the internal format that we store three digits. These digits are the number of years between the year you want to represent and the year 1700. For example, 1989 -1700 is 289 so YYY would be 289. In the examples in this course, we will use arithmetic functions to convert the year from YYYY to YYY.



M Sample Code

```
SET LCD="01/23/1989"
```

```
S INTD=($E(LCD,7,10)-1700)_$E(LCD,1,2)_$E(LCD,4,5)
```

```
WRITE !,INTD
```

Output: 2890123

VA M PROGRAMMING

Intermediate

Pseudocode for VECS6CR, the Create Routine

The following pages display the various steps in adding records to the file as well as reading and validating the various field values. Read through the pseudocode so that you have a thorough understanding of how the program will function.

<i>Line Labels</i>	<i>Steps</i>
START+1:+2	1. Clear the screen and display a heading.
ASK+1	2. Prompt the user to enter the name. If the user entered an ^, pressed Enter, or the prompt times out, go to the Exit (Step 5).
ASK+2 thru +4	3. If the name is not the correct pattern or is too long or the user typed a ?, issue a helpful message and go to Step 2.
ASK+5	4. Set the OK flag to 1 (assume most records will be accessible for LOCKing). DO the Add subroutine (Step 6) to add a record, then if the file is okay for updates, do the Edit fields subroutine (Step 7) to edit the fields, then go back to Step 2.
EXIT para.	5. The Exit: KILL any local variables and QUIT.
ADD+1	6. The Add subroutine: a. Lock the header node so no one else can edit it at the same time. If the header is already locked, wait 5 seconds. If you timeout, issue a "busy" message, set the OK flag to 0, and QUIT (return to Step 4)
ADD+2	b. If the file does not exist, create the header record with the file name (MAILING LIST), file number (1), last record number (0), nr. of entries (0)
ADD+3	c. Find the next available record number: begin with the last record number we got out of the header node; if there is no data node with that number, you've found a record number to use for this entry. However, if the record number is already in use, try the next number in sequence until you find one that is not in use.
ADD+4	d. Update the header with the new record number as the last record number (piece 3) and add 1 to the number of entries (piece 4)
ADD+5	e. Put the name that was entered into the data node for this record number; unlock the node
ADD+6 and +7	f. WRITE the record number that has been assigned and QUIT
EDITFLDS+1 ADDR+1 and +2	7. The Edit fields subroutine: a. Set all the field variables to "" to start b. LOCK the data node so no one else can LOCK it at the same time. If the node is already locked, wait five seconds. If you timeout, issue a "busy" message and go to the edit exit (Step 8)

VA M PROGRAMMING

Intermediate

ADDR+3	c.	Enter the address:
	(1)	Prompt the user to enter the address. If the user timed out or entered "^", go to the edit exit, Step 8.
ADDR+4 and +5	(2)	If the user pressed the Enter key, skip the rest of the line. If the address does not match the pattern or is too long or the user entered ?, display a helpful message and go back to Step 7c.
CITY+1	d.	Enter the city:
	(1)	Prompt the user to enter the city. If the user timed out or entered "^", go to the edit exit, Step 8.
CITY+2 and +3	(2)	If the user pressed the Enter key, skip the rest of the line. If the city does not match the pattern or is too long or the user entered ?, display a helpful message and go back to Step 7d.
STATE+1	e.	Enter the state:
	(1)	Prompt the user to enter the state. If the user timed out or entered "^", go to the edit exit, Step 8.
STATE+2 and +3	(2)	If the user pressed the Enter key, skip the rest of the line. If the state does not match the pattern or is too long or the user entered ?, display a helpful message and go back to Step 7e.
ZIP+1	f.	Enter the zip code:
	(1)	Prompt the user to enter the zip. If the user timed out or entered "^", go to the edit exit, Step 8.
ZIP+2 and +3	(2)	If the user pressed the Enter key, skip the rest of the line. If the zip does not match the pattern or is too long or the user entered ?, display a helpful message and go back to Step 7f.
PHONE+1	g.	Enter the phone:
	(1)	Prompt the user to enter the phone. If the user timed out or entered "^", go to the edit exit, Step 8.
PHONE+2 thru +4	(2)	If the user pressed the Enter key, skip the rest of the line. If the phone does not match the pattern or is too long or the user entered ?, display a helpful message and go back to Step 7g.
LCD+1	h.	Enter the date:
	(1)	Prompt the user to enter the last contact date. If the user timed out or entered "^", go to the edit exit, Step 8Step 8.
LCD+2 thru +4	(2)	If the user pressed the Enter key, skip the rest of the line. If the last contact date does not match the pattern or is too long or the user entered ?, display a helpful message and go back to Step 7h.
LCD+9	(3)	Convert the date from display format (MM/DD/YYYY) to internal format (YYMMDD).
	8.	The Edit Exit subroutine:
EDEXIT+1	a.	Create the new record by concatenating the fields together separated by "^".
EDEXIT+2	b.	Unlock the node.
EDEXIT+3	c.	KILL any variables and QUIT (return to Step 4).

VA M PROGRAMMING

Intermediate

M Code for VECS6CR, the Create Routine

The following pages show M code for the routine that was explained in the pseudocode on the preceding pages.

If there is M code that you do not understand, go back to the previous screen and review the pseudocode for that portion of the routine.

```
VECS6CR          ;ATL/ACE PROGRAMMER-MAILING LIST SYSTEM ;12/1/90
; ;1
;      VARIABLE LIST
;          RECNR      =      Record number selected
;          NAME       =      Individual's name
;          ADDR       =      Address
;          CITY       =      City
;          STATE      =      State
;          ZIP        =      Zip code
;          PHONE      =      Phone number
;          LCD        =      Last date of contact
;          DTIME      =      Delay time
;          OK         =      Flag used with the LOCK
;                          command OK=1 when file is ok
;                          for updates OK=0 when file is
;                          busy
START ;
W @IOF
W !!,?20,"Create Mailing List entry"
ASK ;
R !,"Enter NAME: ",NAME:DTIME I '$T!(NAME["^"]!(NAME="")) G EXIT
I NAME'?1U.U1", "1U.UP!($L(NAME)>30)!(NAME["?"])
I W !,"Enter the name of the desired entry."
I W " Enter last name,first name.",! G ASK
S OK=1 D ADD,EDITFLDS:OK G ASK
EXIT ;
K NAME,ADDR,CITY,STATE,ZIP,PHONE,LCD,OK,RECNR
Q
ADD ;Create a new record
L ^VECS6G(0):5 I '$T W !,"File is busy, try again later" S OK=0 Q
I '$D(^VECS6G(0)) S ^VECS6G(0)="MAILING LIST^1^0^0"
F RECNR=$P(^VECS6G(0),"^",3)+1:1 I '$D(^VECS6G(RECNR)) Q
S $P(^VECS6G(0),"^",3,4)=RECNR_"^"_$P(^VECS6G(0),"^",4)+1)
S ^VECS6G(RECNR,0)=NAME L
W !,"Record # ",RECNR,!
Q
EDITFLDS;Edit data fields
S (ADDR,CITY,STATE,ZIP,PHONE,LCD)=" "
```

VA M PROGRAMMING

Intermediate

```
ADDR      ;
          L ^VECS6G(RECNR,0):5
          I '$T W !,"File is busy, try again later" Q
          R !,"ADDRESS: ",ADDR:DTIME I '$T!(ADDR["^") G EDEXIT
          I ADDR'="",ADDR'? .ANP!($L(ADDR)>25)!(ADDR["?")
          I W !,"Address must not exceed 25 characters.",!,$C(7) G ADDR
CITY      ;
          R !,"CITY: ",CITY:DTIME I '$T!(CITY["^") G EDEXIT
          I CITY'="",CITY'? .ANP!($L(CITY)>20)!(CITY["?")
          I W !,"City must not exceed 20 characters.",!,$C(7) G CITY
STATE     ;
          R !,"STATE: ",STATE:DTIME I '$T!(STATE["^") G EDEXIT
          I STATE'="",STATE'?2U W !,"State must be a 2 character "
          I W "abbreviation.",!,$C(7) G STATE
ZIP       ;
          R !,"ZIP: ",ZIP:DTIME I '$T!(ZIP["^") G EDEXIT
          I ZIP'="",ZIP'?5N&(ZIP'?9N) W !,"Zip must be five or "
          I W "nine digits, no hyphen",!,$C(7) G ZIP
PHONE     ;
          R !,"PHONE: ",PHONE:DTIME I '$T!(PHONE["^") G EDEXIT
          I PHONE'="",PHONE'?3N1"-3N1"-4N&(PHONE'?3N1"-4N)
          I W !,"Phone must be entered with or without area "
          I W "code, using hyphens.",!,$C(7) G PHONE
LCD       ;
          R !,"LAST CONTACT DATE: ",LCD:DTIME I '$T!(LCD["^") G EDEXIT
          I LCD'="",LCD'?2N1"/"2N1"/"4N W !,"Enter a date in the "
          I W "format of MM/DD/YYYY, where MM and DD are each two "
          I W "digits and YYYY is 4 digits.",!,$C(7) G LCD
          ;
          ; THE DATE IS ACCEPTED IN NORMAL DISPLAY FORMAT MM/DD/YYYY
          ; BUT IS STORED IN INTERNAL FORMAT YYMMDD WITH NO SLASHES.
          ;
          I LCD'="" S LCD=($E(LCD,7,10)-1700)_$E(LCD,1,2)_$E(LCD,4,5)
EDEXIT    ;
          S ^VECS6G(RECNR,0)=NAME_"^"_ADDR_"^"_CITY_"^"_STATE_"^"_ZIP_"^"_PHONE_"^"_LCD
          L
          Q
```

VA M PROGRAMMING

Intermediate

Editing the File

There are a few major differences between editing this file and editing the file from Lesson 5.

In the screens that follow, we'll discuss these differences and review the pseudo code and M code for the edit routine.

Finding an Individual

To be able to find an entry by the individual's name, we are going to have to use \$ORDER to search through the record numbers until we find a name to match:

M Code Sample

LOOKUP ;

F RECNR=0:0 S RECNR=\$O(^VECS6G(RECNR)) Q:RECNR'>0

➡ I \$D(^VECS6G(RECNR,0)),(\$P(^VECS6G(RECNR,0),"^",1)=NAME)

➡ S RECORD=^VECS6G(RECNR,0) Q

I RECNR'>0 W \$C(7)," ??",!

Q

The line LOOKUP+1 above must be typed on one statement. However, it is too long to fit on one line in the lesson. The arrow notations ➡ and ➡ are to show you that we couldn't display the entire statement on one line in the curriculum.

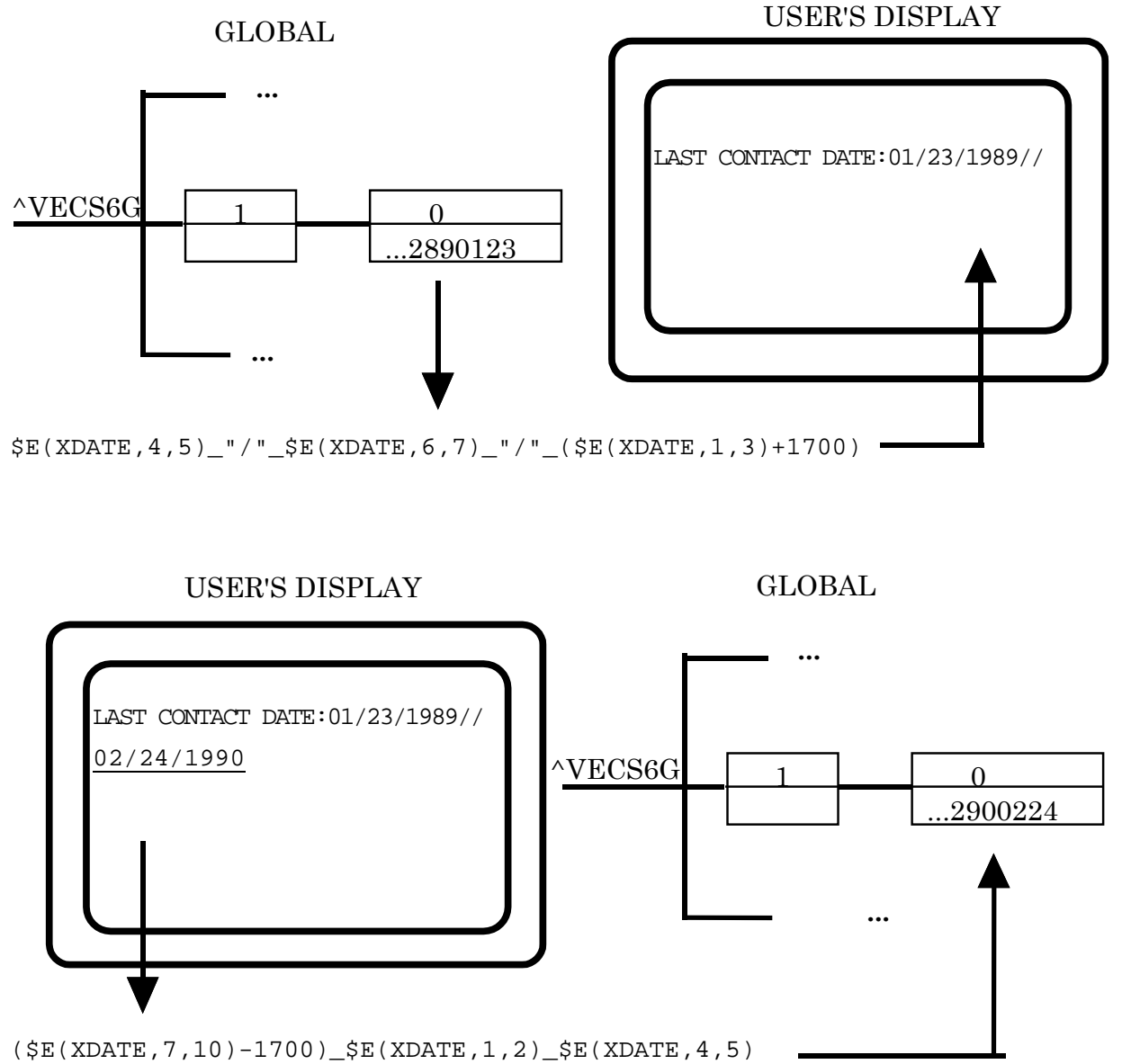
Also, notice that we must use the comma (,) form of the AND operator in the middle of line LOOKUP+1. If a data node does not exist after the \$ORDER is done (which rarely happens but it might if there was a problem on the disk), you do not want to continue and try to take a piece of a nonexistent node.

Date Conversion

You will also notice when you look at the code that we will have to convert the date from internal format to display format for editing and from display format to internal format for storage.

VA M PROGRAMMING

Intermediate



VA M PROGRAMMING

Intermediate

Pseudocode for VECS6ED, the Edit Routine

The following pages show an explanation of the pseudocode for the editing routine. It differs from the create routine just discussed in two important ways.

First, no original record need be stored since we are editing existing records.

Second, in editing data, the original data must be displayed if it is present.

<i>Line Labels</i>	<i>Steps</i>
START+1:+2	1. Clear the screen and display a heading.
ASK+1	2. Prompt the user to enter the name. If the user entered an "^", pressed Enter, or the prompt times out, go to the Exit (Step 5).
ASK+2 thru +4	3. If the name is not the correct pattern or is too long or the user typed a "?", issue a helpful message and go back to Step 2.
ASK+5	4. Do the Lookup subroutine (Step 6) to add a record, then do the Edit fields subroutine (Step 7) to edit the fields only if there is a record to edit for the name entered in the file, then go back to Step 2.
EXIT para.	5. The Exit: KILL any variables and QUIT.
LOOKUP+1	6. The Lookup subroutine: a. Using a loop that starts with RECNR=0 (bypass the header node), get the next record number. If the current record number has an entry in ^VECS6G(RECNR,0) and its name matches the name the user entered, transfer the data to RECORD and stop the loop.
LOOKUP+2	b. If there are no matches on the name (RECNR is null which gives a truth value of 1 to RECNR'>0), sound the bell, and display "??"
LOOKUP+3	c. QUIT the subroutine (return to Step 4).
	7. The Edit fields subroutine:
EDITFLDS+1 and +2	a. LOCK the data node so no one else can LOCK it at the same time. If the node is already LOCKed, wait 5 seconds. If you timeout, issue a "busy" message and go to Edit exit (Step 8)
EDITFLDS+3 thru +6	b. Put each piece of RECORD in a separate variable (name, address, city, state, zip, phone, last contact date).
EDITFLDS+10	c. If there is a date stored, convert it from internal format to display format.
EDITFLDS+11	d. If there is no date, make the display date null (otherwise you'd just have "/"'s displayed)
NAME+1	e. Edit the name: (1) Prompt the user for the name. If there is already data in the field, show it with "/" after it. Accept the user's input into a temporary variable.

VA M PROGRAMMING

Intermediate

NAME+2	(2)	If the user times out or enters an "", go to the Edit Exit (Step 8)
NAME+3 thru +5	(3)	If the user presses the Enter key, skip the rest of the line. If the name does not match the pattern or the name is too long or the user enters a "?", display a helpful message and go back to Step 7e.
NAME+6	(4)	If the data is ok, transfer the value of the temporary variable to the name variable.
	f.	Edit the address:
ADDR+1	(1)	Prompt the user for the address. If there is already data in the field, show it with "/" after it. Accept the user's input into a temporary variable.
ADDR+2	(2)	If the user times out or enters an "", go to the Edit Exit (Step 8).
ADDR+3 and +4	(3)	If the user presses the Enter key, skip the rest of the line. If the address does not match the pattern or the address is too long or the user enters a "?", display a helpful message and go back to Step 7f.
ADDR+5	(4)	If the data is ok, transfer the value of the temporary variable to the address variable.
	g.	Edit the city:
CITY+1	(1)	Prompt the user for the city. If there is already data in the field, show it with "/" after it. Accept the user's input into a temporary variable.
CITY+2	(2)	If the user times out or enters an "", go to the Edit Exit (Step 8).
CITY+3 and +4	(3)	If the user presses the Enter key, skip the rest of the line. If the city does not match the pattern or the city is too long or the user enters a "?", display a helpful message and go back to Step 7g
CITY+5	(4)	If the data is ok, transfer the value of the temporary variable to the city variable.
	h.	Edit the state:
STATE+1	(1)	Prompt the user for the state. If there is already data in the field, show it with "/" after it. Accept the user's input into a temporary variable.
STATE+2	(2)	If the user times out or enters an "", go to the Edit Exit (Step 8).
STATE+3 and +4	(3)	If the user presses the Enter key, skip the rest of the line. If the state does not match the pattern or the state is too long or the user enters a "?", display a helpful message and go back to Step 7h
STATE+5	(4)	If the data is ok, transfer the value of the temporary variable to the state variable.
	i.	Edit the zip:
ZIP+1	(1)	Prompt the user for the zip. If there is already data in the field, show it with "/" after it. Accept the user's input into a temporary variable.
ZIP+2	(2)	If the user times out or enters an "", go to the Edit Exit (Step 8).

VA M PROGRAMMING

Intermediate

ZIP+3 and +4	(3)	If the user presses the Enter key, skip the rest of the line. If the zip does not match the pattern or the zip is too long or the user enters a "?", display a helpful message and go back to Step 7i.
ZIP+5	(4)	If the data is ok, transfer the value of the temporary variable to the zip variable.
	j.	Edit the phone:
PHONE+1	(1)	Prompt the user for the phone. If there is already data in the field, show it with "/" after it. Accept the user's input into a temporary variable.
PHONE+2	(2)	If the user times out or enters an "", go to the Edit Exit (Step 8).
PHONE+3 thru +5	(3)	If the user presses the Enter key, skip the rest of the line. If the phone does not match the pattern or the phone is too long or the user enters a "?", display a helpful message and go back to Step 7j.
PHONE+6	(4)	If the data is ok, transfer the value of the temporary variable to the phone variable.
	k.	Edit the date of last contact:
LCD+1	(1)	Prompt the user for the date of last contact. If there is already data in the field, show it with "/" after it. Accept the user's input into a temporary variable.
LCD+2	(2)	If the user times out or enters an "", go to the Edit Exit (Step 8).
LCD+3 thru +5	(3)	If the user presses the Enter key, skip the rest of the line. If the date of last contact does not match the pattern or the date of last contact is too long or the user enters a "?", display a helpful message and go back to Step 7k.
LCD+10	(4)	If there is a date in the temporary variable then the user entered a new date so convert it to internal format and store it in the date of last contact variable.
LCD+11	(5)	If no date has been entered, convert the original date back to display format.
	8.	The Edit Exit subroutine: :
EDEXIT+1	a.	Recreate the record by concatenating the variables together separated by "^"s
EDEXIT+2	b.	UNLOCK the node, KILL any variables and QUIT (return to Step 4).
EDEXIT+3		



VA M PROGRAMMING

Intermediate





M Code for VECS6ED, the Edit Routine

The following pages show the M code for VECS6ED, the edit routine, which corresponds to the pseudocode for VECS6ED on the preceding pages.

If there is M code that you do not understand, go back to the pseudocode on the preceding pages and review the pseudocode for that portion of the routine.

The lines LOOKUP and EDEXIT in the routine below must be typed on one statement. However, it is too long to fit on one line in the lesson. The arrow notations  and  are to show you that we couldn't display the entire statement on one line in the curriculum.

M Code Sample

```
VECS6ED ;ATL/ACE PROGRAMMER-MAILING LIST SYSTEM ;12/1/90
; ;1
;          VARIABLE LIST
;          NAME, ADDR, CITY, STATE, ZIP, PHONE, LCD...
;          same as CReate routine
;          RECNr      =      Record number assigned in CReate
;          INDATA=     Temporary variable used to hold
;                      entry made by user before it
;                      has been validated
;          XDATE      =      Holds the display format of the date
;          RECORD=     Copy of the contents of the data node
START   ;
W @IOF
W !!,?20,"Edit Mailing List entry"
ASK     ;
R !!,"Enter NAME: ",NAME:DTIME I '$T!(NAME["^"]!(NAME="")) G EXIT
I NAME'?1U.U1","1U.UP!($L(NAME)>30)!(NAME["?"])
I W !,"Enter the name of the desired entry. "
I W "Enter last name,first name.",! G ASK
D LOOKUP,EDITFLDS:RECNr>0 G ASK
EXIT    ;
K NAME,ADDR,CITY,STATE,ZIP,PHONE,LCD,RECNr,INDATA,XDATE
K RECORD
Q
LOOKUP  ;
F RECNr=0:0 S RECNr=$O(^VECS6G(RECNr)) Q:RECNr'>0 
 I $D(^VECS6G(RECNr,0)),($P(^VECS6G(RECNr,0),"^",1)=NAME) 
 S RECORD=^VECS6G(RECNr,0) Q
I RECNr'>0 W $C(7)," ??",!
Q
EDITFLDS ;Edit data fields
L ^VECS6G(RECNr,0):5
I '$T W !,"File is busy, try again later" Q
S NAME=$P(RECORD,"^",1),ADDR=$P(RECORD,"^",2)
S CITY=$P(RECORD,"^",3)
S STATE=$P(RECORD,"^",4),ZIP=$P(RECORD,"^",5)
```

VA M PROGRAMMING

Intermediate

```
S PHONE=$P(RECORD,"^",6),LCD=$P(RECORD,"^",7)
;
; SINCE THE DATE IS STORED IN INTERNAL FORMAT YYYYMMDD,
; WE MUST TRANSFORM IT INTO DISPLAY FORMAT MM/DD/YYYY.
I LCD="" S XDATE=$E(LCD,4,5)_"_"$E(LCD,6,7)_"_"($E(LCD,1,3)+1700)
I LCD="" S XDATE=""
NAME
;
W !,"NAME: ",,$S(NAME="":",1:NAME_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
I INDATA="" ,INDATA'?1U.U1", "1U.UP!($L(INDATA)>30)!(INDATA["?")
I W !,"Enter the name of the desired entry. "
I W "Enter last name,first name.",!,$C(7) G NAME
S:INDATA="" NAME=INDATA
ADDR
;
W !,"ADDRESS: ",,$S(ADDR="":",1:ADDR_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
I INDATA="" ,INDATA'? .ANP!($L(INDATA)>25)!(INDATA["?") W !,"Address must "
I W "not exceed 25 characters.",!,$C(7) G ADDR
S:INDATA="" ADDR=INDATA
CITY
;
W !,"CITY: ",,$S(CITY="":",1:CITY_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
I INDATA="" ,INDATA'? .ANP!($L(INDATA)>20)!(INDATA["?") W !,"City must not "
I W "exceed 20 characters.",!,$C(7) G CITY
S:INDATA="" CITY=INDATA
STATE
;
W !,"STATE: ",,$S(STATE="":",1:STATE_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
I INDATA="" ,INDATA'?2U W !,"State must be a 2 character "
I W "abbreviation.",!,$C(7) G STATE
S:INDATA="" STATE=INDATA
ZIP
;
W !,"ZIP: ",,$S(ZIP="":",1:ZIP_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
I INDATA="" ,INDATA'?5N&(INDATA'?9N) W !,"Zip must be five or "
I W "nine digits, no hyphen",!,$C(7) G ZIP
S:INDATA="" ZIP=INDATA
PHONE
;
W !,"PHONE: ",,$S(PHONE="":",1:PHONE_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
I INDATA="" ,INDATA'?3N1"-3N1"-4N&(INDATA'?3N1"-4N) W !,"Phone must "
I W "be entered with or without area code, using "
I W "hyphens.",!,$C(7) G PHONE
S:INDATA="" PHONE=INDATA
LCD
;
W !,"LAST CONTACT DATE: ",,$S(XDATE="":",1:XDATE_"/") R INDATA:DTIME
I '$T!(INDATA["^") G EDEXIT
I INDATA="" ,INDATA'?2N1"/2N1"/4N W !,"Enter a date in the "
I W "format of MM/DD/YYYY, where MM, DD are each two "
I W "digits and YYYY is four digits.",!,$C(7) G LCD
; AGAIN, THE DATE IS DISPLAYED IN DISPLAY FORMAT MM/DD/YYYY BUT STORED
; IN INTERNAL FORMAT YYYYMMDD WITH NO SLASHES. IF THE DATE HAS BEEN
; CHANGED, THE TRANSFORMATION TO INTERNAL FORMAT IS DONE ON THE VARIABLE
; INDATA--IF IT HAS NOT BEEN CHANGED, XDATE IS USED.
S:INDATA="" LCD=($E(INDATA,7,10)-1700)_$E(INDATA,1,2)_$E(INDATA,4,5)
S:INDATA="" LCD=($E(XDATE,7,10)-1700)_$E(XDATE,1,2)_$E(XDATE,4,5)
```

VA M PROGRAMMING


Intermediate

```
EDEXIT ;
S $P(^VECS6G(RECNR,0),"^",1,7)=NAME_"^"_ADDR_"^"_CITY
  _"^"_STATE_"^"_ZIP_"^"_PHONE_"^"_LCD
L
Q
```


Deleting Records

For this lesson, we have taken the delete routine from Lesson 5.

The line LOOKUP in the routine below must be typed on one statement.

However, it is too long to fit on one line here in the lesson. The arrow notations and  are to show you that we couldn't display the entire statement on one line in the curriculum.

M Code Sample

```
VECS6DEL;ATL/ACE PROGRAMMER-MAILING LIST SYSTEM [ 08/31/92 6:09 AM ]
; ;1
START ;
W @IOF
W !!,?20,"Delete Mailing List entry"
NAME ;
S MISSING=0
R !!, "Enter NAME: ",NAME:DTIME I '$T!(NAME["^")!(NAME="") G EXIT
I NAME'?1U.A1", "1U.AP!($L(NAME)>30)!(NAME["?")
I W !, "Enter the name of the new entry."
I W " Enter last name,first name",! G NAME
D LOOKUP D DELETE:MISSING'=1 G NAME
EXIT ;
K NAME,ANS,MISSING
Q
LOOKUP ;
F RECNR=0:0 S RECNR=$O(^VECS6G(RECNR)) Q:RECNR'>0
   I $D(^VECS6G(RECNR,0)),($P(^VECS6G(RECNR,0),"^",1)=NAME) Q
I RECNR'>0 S MISSING=1
Q
DELETE ;
R !!, "Are you sure? NO//",ANS:DTIME
I '$T!(ANS["^") G DELEXIT
I (ANS="Y")!(ANS="YES") K ^VECS6G(RECNR)
I S $P(^VECS6G(0),"^",4)=$P(^VECS6G(0),"^",4)-1
I W !, "Name deleted!"
E W !, "Name not deleted!"
Q
DELEXIT ;
Q
```

VA M PROGRAMMING

Intermediate

Printing the Output

The major difference between the print routine in Lesson 5 and this print routine is the use of subroutines to access the exact device (in this lesson that means terminal or printer) you will be using to display the report.

Devices

Input/output devices (terminals, printers, disks, etc.), which are attached to a computer system, are assigned identifiers, usually decimal numbers (the exact identifier is implementation specific).

The OPEN Command

To access a device, you must first establish ownership. Basically, only one partition can "own" a device at one time. Ownership is established by using the OPEN command.

In the first M code sample, we want to open device 100.

In the second M code sample, 100 is the device number and 5 is the number of seconds to wait. If the device is "owned" by someone else and we timeout, \$TEST is set to 0.

M Code Sample

```
OPEN 100
```

where 100 is a device number

```
OPEN 100::5
```

```
I '$T W !," .....IN USE !",$C(7) G OPEN
```

where 100 is the device number and 5 is the number of seconds to wait. If the device is "owned" by someone else and we timeout, \$TEST is set to 0.

VA M PROGRAMMING

Intermediate

The USE Command

To actually use a device, however, you must also specify the USE command.

M Code Sample

```
USE 100
```

The CLOSE Command

Input or output from subsequent READ or WRITE commands will go to the device that is in use until you close the device with the CLOSE command or until a USE command is issued to select another device.

M Code Sample

```
CLOSE 100
```

This may only be used by Kernel or through Kernel-supplied utilities:
Commands: OPEN, CLOSE, USE (with parameters)

The \$IO Special Variable

When you logon your terminal, a special variable called \$IO (can be abbreviated to \$I) is set automatically which records the device number of the device you log on with. This device is called the principal device. M does an OPEN and USE automatically for you; all input/output will be directed to this principal device until you specify a different device with the USE command.

If you change devices, \$IO is set to reflect the device currently in use.

Device Control in the VA

Since switching between different devices is a normal part of VA applications, in VA *VISTA* programming, special system-wide variables are used to keep track of which devices are being used and what their associated characteristics are. These variables are:

IO	Hardware name of the last selected input/output device; initially set to the value of \$IO (the current device number)
IO(0)	Hardware name of the input/output device you started out with; initially set to default of \$I; this variable is never altered since it represents your original device

VA M PROGRAMMING

Intermediate

IOF	Character or sequence that clears the screen
IOSL	Length of the device; that is, that is length of screen or paper
IOM	Width of the device; that is, width of the screen or paper
IOST	Device type

For our system, we will not be using these variables directly. Instead, we will make calls to Kernel device-handling routines, which are commonly used for device switching in VA systems.

Pseudocode for VECS6PR, the Print Routine

The following table explains the pseudocode for the various steps of the printing routine.

<i>Line Labels</i>	<i>Steps</i>
START+1	1. Set the page counter to 1 and the user's response at the end of a page to null.
START+2	2. Do the Open subroutine (Step 12). If the user entered an "^" to device number, Kernel routine ^%ZIS would have set a flag to reflect this (in the Open subroutine), and we will go to the Exit (Step 8).
START+3	3. Use the device. From this point on, all output will go to the current device.
START+4	4. Do the Heading subroutine (Step 9),
START+5	5. Do the Order subroutine (Step 10).
START+6	6. Do the Close subroutine (Step 13).
START+7	7. Have the screen pause at the end of the report.
EXIT+1 and EXIT+2	8. The Exit : KILL any variables and QUIT.
HEADING+1 and +2	9. The Heading subroutine: a. If this is not the first page and the device type begins with "C-" (CRT device), display the message to press Enter to continue. QUIT this subroutine if the prompt times out or the user enters an ^ to the prompt.
HEADING+3	b. Clear the screen.
HEADING+4	c. Write the heading.
HEADING+5	d. Add 1 to the page counter.
HEADING+6	e. QUIT the subroutine (return to the end of Step 4)

VA M PROGRAMMING

Intermediate

- ORDER+1 10. The Order subroutine:
- a. Using a loop with the record number starting at 0 (bypass the header node), get the next record number. If there are no more records in the file, stop the loop. Do the Display subroutine (Step 11). If the user has entered an "^" at the end of a page, stop the loop.
- ORDER+2 b. QUIT the subroutine (return to the end of Step 5).
- DISPLAY+1 11. The Display subroutine:
- a. If there is no record for this record number, QUIT the subroutine (return to Step 10a).
- DISPLAY+2 b. Put the contents of the global node into a local variable (it's not efficient to go out to the global frequently).
- DISPLAY+3 c. Write line 1.
- and +4
- LINE2+1 d. Write line 2.
- e. Write line 3:
- LINE3+1 (1) Write the city (piece 3), state (piece 4), and zip (piece 5)
- thru +3 (2) Put the date (stored in internal format) into a variable.
- LINE3+6 (3) If no date was stored, put null in the display format date variable (that way you don't get "/" printed).
- LINE3+7 (4) If there is a date, convert it into display format.
- (5) Display the date
- LINE3+8
- LINE3+9 f. If your current position vertically (\$Y) plus 2 two lines (for a margin) isare greater than the screen or paper length (IOSL), do the Heading subroutine (Step 9).
- OPEN+1 12. The Open subroutine:
- a. Set a Kernel variable %ZIS="M" which will prompt the user to enter a margin width. DO the Kernel routine ^%ZIS which will prompt the user to enter the device and handle device switching. If the user enters an "^" to device, ^%ZIS will set the variable POP=1.
- OPEN+2 b. QUIT the Open subroutine and return to Step 5.
- CLOSE+1 13. The Close subroutine
- CLOSE+2 a. DO the Kernel routine that closes the current device: ^%ZISC
- b. QUIT the Close subroutine (return to Step 6).

VA M PROGRAMMING



Intermediate

M Code for VECS6PR, the Print Routine.

The following pages show the M code for VECS6PR, the print routine.

If there is M code that you do not understand, go back to the pseudocode on the preceding pages and review the pseudocode for that portion of the routine.

The line ORDER in the routine below must be typed on one statement.

However, it is too long to fit on one line here in the lesson. The arrow notations  and  are to show you that we couldn't display the entire statement on one line in the curriculum.

```
VECS6PR ;
;          VARIABLE LIST
;
;          RECNr.....RECNr number assigned in CReate
;          PAGE.....Page number counter
;          ANS.....response to prompt
;          NODE.....contents of ^VECS6G(RECNr,0) node
;          OUT.....holds response: during display of
;                   multiple pages of output, user can press
;                   Enter to continue or ^ to stop display
;          POP.....holds response: when user is asked to
;                   enter a device number, they can enter
;                   a number, press Enter, or ^ to quit.  If
;                   they enter ^, POP gets set to 1
;          XDATE.....date in display format
;          LCD.....date in internal format
;          IN.....input from the device prompt
;
START      ;
S PAGE=1,OUT=""
D OPEN G EXIT:POP
U IO
D HEADING
D ORDER
D CLOSE
R !,"Press Enter to go back to the menu",ANS:DTIME
EXIT      ;
K RECNr,PAGE,ANS,NODE,OUT,POP,LCD,XDATE,IN
Q
HEADING ;
I PAGE>1,($E(IOST,1,2)="C-")
I R !,"Press Enter to continue or ^ to exit",OUT:DTIME Q:'$T!(OUT["^")
W @IOF
W !!,?30,"MAILING LIST LISTING",?60,"PAGE ",PAGE
S PAGE=PAGE+1
Q
```

VA M PROGRAMMING

Intermediate

```
ORDER      ;
           F RECNR=0:0 S RECNR=$O(^VECS6G(RECNR)) Q:RECNR'>0
           D DISPLAY Q:OUT[ "^"
           Q
           DISPLAY ;LINE 1
           Q:'$D(^VECS6G(RECNR,0))
           S NODE=^VECS6G(RECNR,0)
           W !!!,$P(NODE,"^",1)
           W ?40,"PHONE: ",$P(NODE,"^",6)
LINE2      ;
           W !,$P(NODE,"^",2)
LINE3      ;
           W !,$P(NODE,"^",3),?$X+2
           W $P(NODE,"^",4),?$X+2
           W $P(NODE,"^",5)
           ; WE WANT THE DATE PRINTED IN DISPLAY FORMAT MM/DD/YYYY
           ; NOT INTERNAL FORMAT YYMMDD
           S LCD=$P(NODE,"^",7)
           I LCD="" S XDATE=""
           I LCD="" S XDATE=$E(LCD,4,5)_"_"$E(LCD,6,7)_"_"($E(LCD,1,3)+1700)
           W ?40,XDATE
           D:$Y+2>IOSL HEADING
           Q
OPEN        ;
           S %ZIS="M" D ^%ZIS
           Q
CLOSE       ;
           D ^%ZISC
           Q
```

Limitations

The following are some of the limitations you will find with the sample system:

1. There is still a lot of redundant code. The create and edit options use the same validation.
2. In the edit, we can't delete a name or a field entry.
3. You can accidentally add a duplicate name: the second occurrence of the same name is stored separately without checking to see if the same name exists.
4. In the edit, you will find only the first occurrence of a matching name; that means you can accidentally edit the wrong occurrence of a name.

All of these limitations will be addressed and resolved in future lessons.

VA M PROGRAMMING

Intermediate

Assignment (Optional)

In General

1. Follow the VA Programming Conventions as illustrated in this lesson.
2. Use only the techniques, commands, and features described in this lesson or the previous lessons.
3. Save the routine(s) on a test system under the namespace assigned to you by your local IRM Chief or IT Manager. For this lesson, use the following suffix pattern:

VEC6A	for the menu routine
VEC6B	for the create routine
VEC6C	for the edit routine
VEC6D	for the delete routine
VEC6E	for the print routine

Important

You will find that it is possible to use your routines from Lesson 5 as the basis for the assignment for this lesson. Make sure, however, that you change the namespaces of each routine to reflect that this is a Lesson 6 assignment.

You will need to create a global file. In your routine, use the namespace pattern below:

VEC6G

where: G stands for Global

Assignment Specifications

You are to redesign and reimplement the personnel system you first developed in Lesson 5. You should now follow the Sample System in this lesson. The global file should be structured with a file header node and use internal record numbers; the global file must contain the following fields (each field should be validated according to the format below):

Field	Format
Name	Last name,First name (all CAPS; no space before or after the comma)
Address	No more than 25 characters
City	No more than 20 characters
State	2-letter code
Zip	5 numeric digits
Social security number	nnn-nn-nnnn (use hyphens only)

VA M PROGRAMMING

Intermediate

Department	a code: 01, 02, 03, or 04 are the only codes that are acceptable (make sure the user types in the leading zero)
Job Title	No more than 20 characters
Home phone number	nnn-nnn-nnnn (use area code and hyphens)
Employment date	display in mm-dd-yyyy (use hyphens; sample shows /'s but you are to use -'s) mm is month number dd is day number yyyy is year store as yyymmdd (no hyphens)

NOTE: Please make sure that your system prompts for the above fields in the order shown. Also, follow each prompt with a colon (:). For example, your prompt for the name field might be: **Enter Name** | _ | : | _ | *where | _ | indicates a space.*

Your system should have the following functions:

- Menu which "drives" the entire system: make it "generic"
- Create routine for new entries:
 - Use LOCK where appropriate
- Edit routine to edit entries:
 - Make sure to show the default responses
 - Use LOCK where appropriate
- Print routine to print ALL FIELDS in the file:
 - Design the output in whatever format you choose
 - Allow the user to select a different device for the output

All prompts should be timed and escapable. The user should be able to type a ? and get help.