Chapter 9

File Handling

1 Review of File Handling

2 The Phone List Program

3 Error Handling

G O A L S

After working through this chapter, you will be able to:

Understand and use the concepts associated with computer databases.

Manipulate random access files.

Program a simple database.

Integrate error-handling routines with programs.

OVERVIEW

This chapter covers handling data stored in a file. A random access file is used to store names and phone numbers. The information in this chapter is easily adapted to almost any problem involving stored data.

You will also learn in this chapter how to anticipate the possibility of run-time errors when you write code, so that your program can respond to (or "handle") errors that occur when it runs. Programs that can handle run-time errors are more reliable than those that cannot. Each kind of run-time error that a program handles is one fewer source of unexpected crashes.

Review of File Handling

You have already had experience by this point with handling files. In the programs you've built so far, though, you have been focused on finishing the work at hand. You may not have thought about what you were learning about file handling as a way of working. This section reviews what you have learned so far about file handling and brings all the information into one place.

Types of Files

There are three kinds of files that may be opened and accessed in a Visual Basic program: random, sequential, and binary. In a random file, the data is organized into records. A record is a complete set of data for a single entry. Each record uses a predetermined amount of memory declared when the file is opened. For instance, a person's first and last name, social security number, age, and address make up a single record in a file.

A sequential file is used as if it is printed output. The command that prints a line to the printer or to a form can print a line to a sequential file. The same command that collects information from the keyboard is used to collect information from a sequential file. Records in a sequential file are arranged like songs on a cassette tape: one after another.

A binary file is a file arranged to give byte-by-byte access to the user. In a binary file, information is recorded in its most primitive form: the byte.

Open Statement

Before you read from or write to a file, you must open it. You do so with the **Open** statement. This statement tells Visual Basic whether you will use the file as a random, sequential, or binary file. Visual Basic then tells Windows to open the file on behalf of your program. Windows attempts to find or create the file you wish to open. If it succeeds, it sets up internal data structures that record the fact that your program has opened the file, and that keep track of the current position in the file.

Here's an **Open** statement:

```
Open "c:\plist\phone.dat" For Random Access Write As #1 Len = 55
```

After the word "Open," you put the full pathname of the file you want to open. The pathname is listed in the statement or represented by a string variable. After the word "For," you put the mode of file access. This is either one of the keywords (**Random** or **Binary**), indicating the type of file, or it is one of three other options if the file is a sequential file.

Mode	*Description*
Random	Files organized as records
Binary	Files organized as bytes
Input	For using a sequential file for input operations
Output	For using a sequential file for output operations
Append	For adding information to a sequential file

Next, you indicate the type of access. There are three options:

- Read: You can open the file but cannot make any changes.
- Write: You can create the file or modify it if it already exists.
- Read Write: You can perform either type of operation; this mode applies to random and binary files as well as to sequential files opened for the Append operation.

The **As #** section of the **Open** statement assigns a file number to the file so that other statements may refer to the file by its number rather than its pathname. The **Len = *n*** section designates the number of characters per record for a file of the random type.

This **Open** statement:

```
Open "c:\plist\phone.dat" For Random Access Write As #1 Len = 55
```

opens the file **c:\plist\phone.dat** as a random access file, opened for writing (the file is created if it does not exist) as file #1, with a record length of 55 characters.

Put Statement

The **Put** statement writes data to a file opened as a **Random** or **Binary** file. Here is an example of the syntax:

```
Put #1, x, PhoneList(x)
```

Following the word "Put" is the file number. Information will be placed into the file associated with the file number 1, an association made with an **Open** statement.

For **Random** files, following the file number is the record number of the data being put into the file. Assuming that #1 is a **Random** file, the variable *x* indicates that the contents of *PhoneList(x)* are put into the file as record number *x*. As *x* changes, each successive piece of data is put into an appropriate spot in the file.

For **Binary** files, the number following the file number is the byte position of the data being put into the file. It is an offset from the beginning of the file, which is byte 1. Assuming that #2 is a **Binary** file, the

following statement writes an integer to the file starting at the 100th byte of the file:

```
Dim N
'— ...
Put #2, 100, N
```

If you omit *x* from the **Put** statement, each record is inserted at the current position in the file. The current position is the one immediately following the data written by the last **Put** statement, read by the last **Get** statement, or explicitly set via the **Seek** function. Here's the **Put** statement without the middle value:

```
Put #1, , PhoneList(x)
```

Get Statement

The **Get** statement reads data from an file opened as either a **Random** or **Binary** file. This statement matches the **Put** statement and has a similar syntax:

Get #*file number*, *record number*, *variable*

If #*file number* is a **Random** file, then *record number* refers to the position in the file where the record whose number is *record number* is stored. If #*file number* is a **Binary** file, then *record number* refers to a byte offset from the beginning of the file. The data at this location is read from the file and stored in *variable*.

Here's a typical example of the **Get** statement used with a **Random** file #1:

```
Get #1, x, PhoneList(x)
```

This statement fetches the *x*th entry from the file and assigns it to *PhoneList(x)*. Just as in the **Put** statement, if you leave the middle number out, the records are fetched from the file sequentially.

Close Statement

Without any parameters, the **Close** statement closes all open files. To close a specific open file, use the file's number, as in:

```
Close #1
```

This statement tells Visual Basic that you are done using this file number.

Kill Statement

The **Kill** statement deletes a file whose pathname is specified. Here is an example:

```
Kill "c:\plist\phone.dat"
```

If the file doesn't exist, a run-time error occurs. Program execution stops.

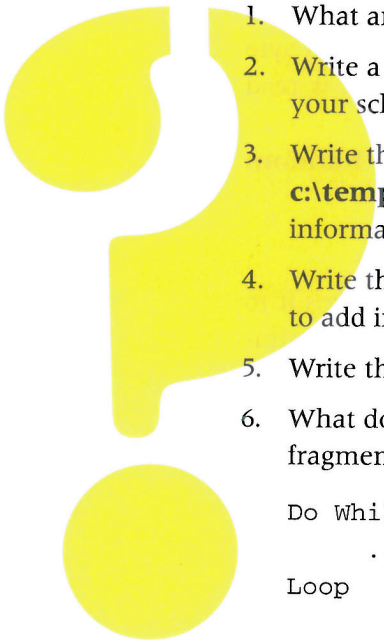
EOF Function

EOF stands for End Of File. This function returns the value of **True** when the **Get** command has *failed* to get information from the file because the file is empty. The syntax for this function is:

EOF(*#file number*)

Be sure to notice that **EOF** returns a value of **True** when **Get** has failed. This means that the program makes one attempt to get information from the file beyond the end of the file.

QUESTIONS AND ACTIVITIES



1. What are the three kinds of data files supported by Visual Basic?
2. Write a record that contains eight pieces of information about your school. Include data such as the name and address.
3. Write the **Open** statement that opens the file named **c:\temp\petname.txt** as a sequential file ready to receive information.
4. Write the **Open** statement that opens the file mentioned above to add information to the end of the file.
5. Write the statement that deletes the file created in question 3.
6. What does **EOF** stand for? Describe the purpose of **EOF** in the fragment below.

```
Do While Not EOF(1)
     ...
Loop
```

The Phone List Program

In this section you will write a program to maintain a list of people's names and phone numbers. A List box is used to display the list.

A file is used to store the information. The path and file name is: **c:\plist\phone.dat**.

The program can

- Save the current list
- Clear an old list
- Open an existing list and load it from a file
- Add names to the list
- Display them sorted by first name, last name, or by phone number

You will use a user-defined variable type to represent the information. The program uses the **Open** statement to open a random access file for reading and writing. The operations it performs on the file's data—saving, clearing, opening, creating, and adding to lists—are representative of what every database program does.

Starting Out

Before you start building the form, you need to learn how to use user-defined data types in Visual Basic. You will learn what user-defined types are, how to declare them, how to declare variables of a user-defined type, and how to use such variables.

A NEW KIND OF DATA TYPE

A record is composed of one or more logically related variables. To relate these variables, you can create a user-defined data type. This type combines one or more variables of various types into a single data type. You specify the name and composition of the data type. After you have defined the data type, you can use it in a program just like one of the predefined types.

Declaring a user-defined type isn't difficult. Here's an example based on the project you are about to create:

```
'—Each record is 55 characters
Type Person
   Last As String * 20
   First As String * 20
   Phone As String * 15
End Type
```

The word "Type" is followed by the name of the new data type, Person. The variables and data types that make up the body of the new data type are listed. The Person type is made of three strings, *Last, First,* and *Phone*. Each item in the body of the type definition is a field.

You can define user-defined types only within a code module. Type definitions appear in the general declarations section of a code module. Any variables you declare or types you define within a code module are global. You can use them in any of the event procedures of a form, as well as in any event procedure of any form or code module in the project.

To open the code module's window, double-click on its name in the Project window. The caption of the code module's window reflects the file name of the module. See Figure 9-1.

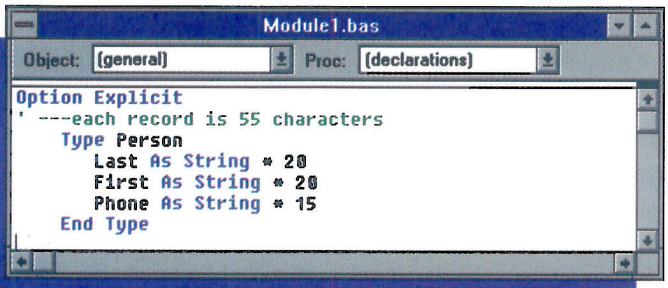


Figure 9-1
Caption of the code module's window reflecting the filename of the code module

ASSIGNING VALUES

You will need to know how to assign values to the fields of a record, and how to use these values in expressions and statements. Assume the following declaration:

```
Dim Aunt As Person
```

To assign the aunt's name and phone number, use the following statements:

```
Aunt.Last = "Johnson"
Aunt.First = "Maudie"
Aunt.Phone = "(708)934-1111"
```

If you declare an array of type Person, you can access the third element of the array using these statements:

```
Dim Relatives(1 To 10) As Person
Relatives(3).Last = "Johnson"
Relatives(3).First = "Tracie"
Relatives(3).Phone = "(708)949-1991"
```

THE WITH STATEMENT – VISUAL BASIC 4

In Version 4 of Visual Basic, the **With** statement simplifies access to the fields of a record. (To check which version you have, select About Microsoft Visual Basic from the Help menu.) For example, assume the following declarations:

```
'–Code module definition of Type
Type Person
   Last As String * 20
   First As String * 20
   Phone As String * 15
End Type
'–Local declaration of variables
Dim Relatives(1 To 10) As Person
Dim PersonNumber As Integer
'–Assign values to the third element as follows
PersonNumber = 3
With Relatives(PersonNumber)
   .Last = "Johnson"
   .First = "Tracie"
   .Phone = "(708)949-1991"
End With
```

Planning the Program

What functions should a phone list program have? The user should be able to record new names and numbers, save the list to permanent storage, recall the list, and list the names and numbers.

One way to bring program development into focus is to plan how the user will interact with the program. You will need a File menu that includes the standard options for handling files:

File
- **New...**
- **Open...**
- **Save**
- **Close**
- ---
- **Exit**

To let users add and list names, how about adding another menu:

Names
    Adding...
    Listing...

When users select Adding from this drop-down menu, you need a way for them to enter new data. The best way is to provide a dialog box for this purpose. The dialog box should open when a user selects the command.

If, instead, the user selects Listing, another menu should open:

Listing...
    Last
    First
    Phone

The commands in this submenu all perform a similar task: each command forms strings from the Person records, then calls a routine that displays that list of strings in a List box. The three menu commands differ only in the order of the record fields that they use to form strings. It's a good idea, then, to make these commands a menu command array. The List box in which the strings are displayed has its Sorted property set to **True**. Different lists are created by rearranging the information displayed. If the first name is first in the display string, the list is ordered by first names.

Creating the Form

The user interface is the menu discussed above. In this section you create a directory for the program files and the data file and set up the form.

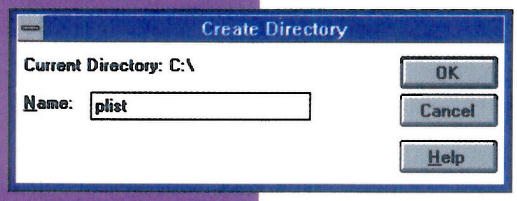


Figure 9-2
Creating a directory

1. Open the Program Manager. Start the File Manager.
2. Create a subdirectory in **c:** named **c:\plist**. See Figure 9-2.
3. Start Visual Basic. If Visual Basic is running, choose New Project from the File menu.
4. Change the caption of the default form to **Phone List**. Change the name of the form to **frmPlist**.
5. Select the form and open the Menu Design window.
6. Enter the caption **&File** and the name **mnuFile**.
7. Click on Next, then the right arrow button to create a submenu item.

8 Enter the captions and names in the table below, clicking on Next between each entry.

Caption	*Name*	*Enabled*
&New...	mnuNew	
&Open...	mnuOpen	
&Save	mnuSave	False
&Close	mnuClose	False
-	mnuHyphen	(a separator)
E&xit	mnuExit	

9 Click on Next, then on the left arrow button.

10 Enter the caption **&Names** with the name **mnuNames**.

11 Click on Next, then on the right arrow button to create a submenu of Names. Enter the following captions and names.

Caption	*Name*	*Enabled*
Adding...	mnuAdd	False
Listing...	mnuList	False

12 Click on Next, then on the right arrow button to create a submenu of Listing. Enter the following captions. Create a control array for these related commands.

Caption	*Name*	*Index*
Last	mnuSort	1
First	mnuSort	2
Phone	mnuSort	3

13 Add a large List box to the form. Change the name of the List box to **lstPhoneNumbers**. Figure 9-3 shows the complete form.

14 Save the project and form files.

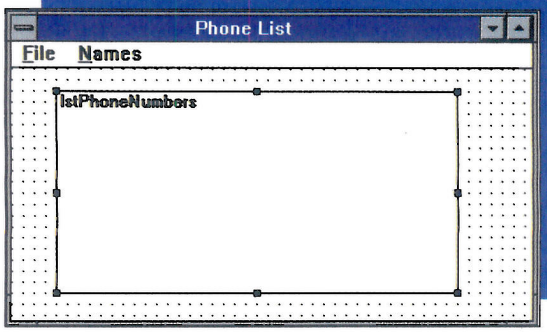


Figure 9-3
Complete form

Creating the Dialog Box

Now create a dialog box to collect a name and phone number.

1. Add a new form to the project. Change the caption of the form to **Name and Phone Number**. Change the name of the form to **frmData.**
2. Add three textboxes to the form. Name the boxes **txtLast**, **txtFirst**, and **txtPhone**. Delete the text from each.
3. Add three labels with captions: **Last Name:**, **First Name:**, and **Phone Number:**.
4. Add a command button with the caption **&OK** and the name **cmdOK**.
5. Add a command button with the caption **&Cancel** and the name **cmdCancel**. The complete dialog box is show in Figure 9-4.

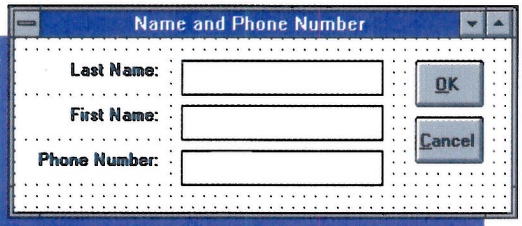


Figure 9-4
Complete dialog box

Declaring Variables

The project has two forms. The two forms communicate with each other through global variables declared in a code module. The variables *Current* and the array *PhoneList*, of type Person, are declared globally. When information is collected from the dialog box, it is put into the array *PhoneList* and *Current* is updated.

The *PhoneList* array, with a capacity of 55 entries, is used to store the information in memory. *Current* indicates the number of entries in the *PhoneList* array. This number is set when information is read from a file, and reset when names are added to the array. The counter is initialized to 0 when the program is executed.

Current is also used to control the display of information in the List box and for writing information into the data file.

To declare the variables:

1. Add a code module to the project.

2 In the general declarations section of the code module, add the following declarations:

```
Type Person
   Last As String * 20
   First As String * 20
   Phone As String * 15
End Type
Global PhoneList(55) As Person
Global Current As Integer
```

3 In the Project window, select frmPlist.

4 In the Form_Load procedure, enter the code to initialize the counter *Current*:

```
Current = 0
```

5 Save the forms, the code module, and the project in **c:\plist**.

Adding Information

Users add information in two steps. First, they click on the Adding command in the start-up form, frmPlist. The code in that procedure tests to see if there is more room in the array and then displays the dialog box, frmData. When all the necessary information has been added to the dialog box, the user clicks OK to transfer the information to the *PhoneList* array.

In the following steps, you create the code that lets the user add data in the way just described:

1 Select frmPlist in the Project window.

2 Click on View Code.

3 Enter the following code in the procedure for mnuAdd:

```
'—Check value of Current to see of array if full
If Current >= 55 Then
   MsgBox "The array is full"
Else
'—Show dialog box to collect data
   frmData.Show
End If
```

The **Show** command makes the dialog box visible.

4 Select frmData in the Project window. Click on View Code.

5 In the cmdOK_Click() procedure, enter the local declarations:

```
'—Declare local variables
Dim Last As String * 20
Dim First As String * 20
Dim Phone As String * 15
```

6 Enter the lines to transfer information from the textboxes to the local variables:

```
'—Read information from textboxes
Last = txtLast
First = txtFirst
Phone = txtPhone
```

7 Test to see if at least the first name and the phone number are not blank. The **Trim$()** function deletes spaces from the strings.

```
'—Must have at least first name and phone number
If Trim$(First) <> "" And Trim$(Phone) <> "" Then
```

8 Enter the lines that increment the counter, *Current*, and add the new information to the array:

```
'—Increment Current and use as a subscript for PhoneList
Current = Current + 1
PhoneList(Current).Last = Last
PhoneList(Current).First = First
PhoneList(Current).Phone = Phone
```

9 Now that information is added to the form, make sure the proper menu commands from frmPlist are enabled. Because the menu commands are defined on the form, the object name, frmPlist, must appear. Enter the lines to turn on menu commands in frmPlist:

```
'—Accessing routines from the frmPlist form.
frmPlist.mnuList.Enabled = True
frmPlist.mnuSave.Enabled = True
frmPlist.mnuClose.Enabled = True
```

10 Enter the lines to blank the textboxes, preparing for the next add operation:

```
'—Blanking textboxes for next add operation.
txtLast = ""
txtFirst = ""
txtPhone = ""
```

11 Enter the lines to hide the dialog box:

```
'—Hide the dialog box
frmData.Hide
```

12 If the user did not enter the minimum data—first name and phone number—set the focus back to the first name:

```
Else
'—Set focus to first name box
   txtFirst.SetFocus
End If
```

13 In the procedure for cmdCancel of frmData enter the following command:

```
frmData.Hide
```

14 This completes the code for frmData.

Saving the Phone List

Assume that names have been added to the *PhoneList*. It's time to create the data file and save the information. To do that, you need to:

- Open the file
- Set up a loop from 1 to *Current*
- Put each record in the file
- Close the file

Follow these steps to enter the code.

1 Select frmPlist and enter the following lines in the procedure for mnuSave:

```
Dim Entry As Integer
'—Open file for Random Access Write
Open "c:\plist\phone.dat" For Random Access Write As #1 Len = 55
'—Write Current entries of the array into the file
For Entry = 1 To Current
     Put #1, Entry, PhoneList(Entry)
Next Entry
'—Close the file
Close
```

2 Save the forms, code module, and project file.

Opening an Existing File

To open an existing file:

- Check to see if there is an active list
- If not, set subscript variable to one
- Open the file
- Loop while the file is not empty
  - Get information from the file
  - Load information into the *PhoneList* array
  - Increment the subscript
- Close the file
- Set the value of *Current*
- Enable various menu items

Follow these steps to create the mnuOpen procedure:

1. Select frmPlist and enter the following lines in mnuOpen_Click(). First enter the declaration of a local variable.

```
'—Variable to keep track of entry number
Dim Entry As Integer
```

2. If *Current* is anything but 1, a file has already been loaded or a new one created. Enter the lines that handle this:

```
If Current <> 0 Then
   '—Don't open the file if the list is already loaded.
   MsgBox "There is already a list loaded."
Else
```

3. Enter these lines to open the data file and transfer information into the array:

```
'—Otherwise, set entry to one and collect data.
Entry = 1
Open "c:\plist\phone.dat" For Random Access Read As #1 Len = 55
Do While Not EOF(1)
   Get #1, , PhoneList(Entry)
   Entry = Entry + 1
Loop
Close
```

4 Because **EOF** (End Of File) is true only after **Get** has failed, the value of *Entry* is 2 beyond the end of the list. Enter the code that sets *Current* and enables some menu items:

```
    '—Entry is two beyond the end of the list
    Current = Entry - 2
    '—Enable menu items in frmPlist
    mnuSave.Enabled = True
    mnuClose.Enabled = True
    mnuList.Enabled = True
    mnuAdd.Enabled = True
End If
```

5 Save the forms, code module, and project file.

Coding the New Command

Selecting New closes and deletes any existing file. Before the file is closed, you should prompt the user to be sure that's what the user wants to do. Use an InputBox to get confirmation.

If a file doesn't exist, deleting it causes a run-time error. The **On Error Resume Next** statement will trap the error generated by trying to kill a nonexistent file and resume execution with the statement following the one that caused the error.

Set *Current* to 0, indicating there are no valid entries in the array. The new file is opened and closed, creating a directory entry and a file of length 0.

1 Select frmPlist and enter the following lines for mnuNew_Click(). First enter the declaration of a local variable. Use an InputBox to ask the user to confirm deleting the old list of names. The **Left$(str,number)** function copies the first *number* characters from the left end of the string.

```
'—Check with user to see if this is what they want
Dim answer As String
answer = InputBox("Do you want to clear the file?", "New...", "No")
If Left$(answer, 1) = "Y" Or Left$(answer, 1) = "y" Then
```

If the user enters anything that starts with "y" or "Y", the procedure ends.

2 If the file doesn't exist, the **On Error** statement prevents a run-time error. If the file is found, it is killed and *Current* is reset to 0:

```
'—If file doesn't exit, this will trap the error.
On Error Resume Next
'—Erase file from disk
Kill "c:\plist\phone.dat"
Current = 0   ' Number of elements in list
Open "c:\plist\phone.dat" For Random Access Write As #1 Len = 55
Close
```

3 Enter the code to clear the List box of names and numbers. Enter the lines to enable various menu items:

```
        lstPhoneNumbers.Clear
        mnuSave.Enabled = True
        mnuList.Enabled = True
        mnuAdd.Enabled = True
    End If
```

4 Save the forms, code module, and project file.

Coding the Close Command

The **Close** procedure clears the List box and sets *Current* to 0, effectively clearing the array of information. A number of menu items are disabled. Once the file is closed, Save, Close, Adding, and Listing become inappropriate actions.

To code the **Close** command, open mnuClose_Click() and enter this code:

```
lstPhoneNumbers.Clear
Current = 0
mnuSave.Enabled = False
mnuClose.Enabled = False
mnuAdd.Enabled = False
mnuList.Enabled = False
```

Writing the Display Routines

The program displays information contained in the *PhoneList* array in the List box lstPhoneNumbers. The Sorted property of the List box is set to **True** at design time. By changing the order of the data within each display line, *DispLine*, the overall ordering of the information is changed.

Display the last name first; the List box sorts using the last name. Display the first name first in each line; the List box sorts using the first name. *Entry* keeps track of the subscripts of the information to display. *DispLine* is a string used to assemble the display line for the List box. *Tb* is the tab character. It is used to display the entries in columns.

MnuSort is a menu control array. When clicked, the three menu commands in the Listing menu (Last, First, and Phone) execute the same event procedure, mnuSort()_Click.

1. Select frmPlist and open the cmdSort() procedure.
2. Enter the lines declaring the local variables and setting up the tab character:

```
'—Declare local variables.
'—used as subscript
Dim Entry As Integer
'—used to build display line
Dim DispLine As String
Dim Tb As String
'—Tab character
Tb = Chr$(9)
```

3. Enter the lines to clear the List box and check to see whether the array is empty:

```
'—Clear the listbox.
lstPhoneNumbers.Clear
'—Display list only if there are elements in the array
If Current <> 0 Then
```

4. If the array is not empty, enter the line to set up the loop. There are *Current* entries in the array.

```
For Entry = 1 To Current
```

5 Enter the lines that pick out what menu command is selected. Arrange the display line according to the kind of list chosen.

```
'—Select statement picks out clicked menu item
Select Case index
'—Arrange by last name
   Case 1
      DispLine = PhoneList(Entry).Last & Tb & PhoneList(Entry).First & Tb &
PhoneList(Entry).Phone
'—Arrange by first name
   Case 2
      DispLine = PhoneList(Entry).First & Tb & PhoneList(Entry).Last & Tb &
PhoneList(Entry).Phone
'—Arrange by phone number
   Case 3
      DispLine = PhoneList(Entry).Phone & Tb & PhoneList(Entry).First & Tb &
PhoneList(Entry).Last
End Select
```

6 Enter the line to add the display line to the List box:

```
'—Add line to listbox
lstPhoneNumbers.AddItem DispLine
```

7 Enter the lines to end the loop and handle the **Else** case of the **If** statement from above. These lines execute if the array is empty:

```
   Next Entry
Else
   '—If Current = 0, the array is empty
   MsgBox "The list is empty"
End If
```

8 Run the program. Click on New. Enter names and phone numbers. List the names with each of the three display options. Save the file. Press New. Open the old file. Display the list.

9 Save the form files, the code file, and the project file.

QUESTIONS AND ACTIVITIES

1. Write a definition for a user-defined type to represent the following information:

name of a computer	20 characters
name of processor	10 characters
speed of processor in MHz	integer
megabytes of RAM	integer
megabytes of disk space	integer
cost	currency

2. Declare an array of the computer type declared above. The array should store information for 25 computers.
3. How would you refer to the price of the fourteenth machine in the list created above?
4. Assuming the array above is full, write the code that would display the contents of the array in a List box.
5. What are the three file types?
6. Describe in a few words what each of these statements does:
   Open, **Close**, **Kill**, **Get**, and **Put**.
7. Explain the function of the ...arate underlined part of the following **Open** statement:

```
Open "c:\vb\phone.dat" For Random Access Write As #1 Len = 55
```

8. What are the three parameters of the **Put** statement? Describe each one.
9. In the Phone List program:
   a) Modify to accept a social security number as well as names and phone numbers.
   b) Modify the Listing submenu to include a command to sort the list by social security number. Include the statements necessary in mnuSort().
10. Modify the Phone List program to get a file and pathname from the user.
11. In cmdCancel of frmData, replace frmData. Hide with Unload Me. Run the program and note any differences. Use the Help system to look up Unload and Me.

Error Handling

The Phone List program used a file and pathname permanently written into the code. The MiniEdit program used an InputBox to get the pathname from the user. Programs accessing files should let the user enter file names. Problems arise when the user enters the name of a file that doesn't exist or enters an illegal pathname.

So far, the **On Error Resume Next** statement has been the extent of the error handling. Unfortunately, this allows the program to continue even when a serious error has occurred. The program should not continue without correcting or responding to the error.

Looking at the Options

A run-time error occurs when something interrupts the execution of a program, such as division by 0, an array subscript that is out of bounds, or the use of an illegal pathname. If there is no error trapping, the program stops with an error message. Error trapping means redirecting the flow of execution when errors occur.

The **On Error Resume Next** statement traps an error by executing the statement following the one that caused the error. The action that caused the error is never completed.

The **On Error GoTo** ***line*** statement redirects the flow of execution to an error-handling routine. An error-handling routine is a piece of code that either resolves the error and lets the program resume its normal course or displays information about the error so that it may be corrected.

Flow is redirected by specifying a line number or a line label. A line label is an identifier, up to 40 characters long, followed by a colon. A line label must be the first entry in a line.

When a run-time error occurs, the **Err** function is set to an error code. You can find a list of error codes in Visual Basic Help, under the heading "Trappable Errors". The **Erl** function, unused in this program, returns the number of the line where the error occurred. Displaying the error code as a part of the error-handling routine helps the user or programmer know how to fix the problem.

The **Exit Sub** statement interrupts the flow of program execution and ends the procedure being executed. An error-handler is a piece of code within the procedure, usually at the end of the procedure. The code should not be run if there are no errors. Placed before the error-handling code, the **Exit Sub** statement leaves the procedure before the error-handling code can execute.

Modifying the Open Routine

Use the **On Error GoTo** statement by modifying the mnuOpen routine of the Phone List program. This modification traps illegal file names. You should also modify the routine in a second way, by adding the **InputBox** function to collect the name of the file from the user.

To make these changes:

1. Open the event procedure for mnuOpen. You need to keep the following lines; look through the code until you find the end of this section:

```
Sub mnuOpen_Click ()
'—Variable to keep track of entry number
Dim Entry As Integer
If Current <> 0 Then
   '—Don't open the file if the list is already loaded.
   MsgBox "There is already a list loaded."
Else
   '—Otherwise, set entry to one and collect data.
   Entry = 1
```

2. Enter the lines declaring a string variable for the filename and collecting a filename with an **InputBox$**:

```
Dim FName As String
FName = InputBox$("Enter the complete path name", "Open File")
```

3. Enter the next two lines. The second redirects the flow of the procedure to the code marked ErrorHandler at the end of the procedure:

```
'—Declare variable for message.
Dim Msg
'—Set up error handler.
On Error GoTo ErrorHandler
```

4. The next line is modified from the original routine. Change the hard-coded filename **c:\plist\phone.dat**, to the filename **FName**:

```
Open FName For Random Access Read As #1 Len = 55
```

5 The next lines shown are already part of the mnuOpen routine, skip them:

```
    Do While Not EOF(1)
       Get #1, , PhoneList(Entry)
       Entry = Entry + 1
    Loop
    Close
    '—Entry is two beyond the end of the list
    Current = Entry - 2
    mnuSave.Enabled = True
    mnuClose.Enabled = True
    mnuList.Enabled = True
    mnuAdd.Enabled = True
End If
```

6 The ErrorHandler follows. To avoid executing the error-handling code in a normal completion of the procedure, add the **Exit Sub** statement:

```
Exit Sub
```

7 Enter the lines of the ErrorHandler routine. The first line is the line label, ErrorHandler. The **Select Case** statement, using **Err**, prints an appropriate error message. Once the correct message is displayed, the **Exit Sub** statement ends the procedure without any change to the labels:

```
ErrorHandler:
Select Case Err
   Case 53: Msg = "ERROR 53: That file doesn't exist."
   Case 68: Msg = "ERROR 68: Drive " & Drive & ": not available."
   Case 76: Msg = "ERROR 76: That path doesn't exist."
   Case Else: Msg = "ERROR " & Err & " occurred."
End Select
MsgBox Msg      ' Display error message.
Exit Sub
End Sub
```

8 Run the program. Enter an illegal filename. Use a drive letter that doesn't exist or use an illegal character or length. The error handler prints a message and leaves mnuOpen as if nothing has happened.

9 Save the form files, the code module, and the project file.

QUESTIONS AND ACTIVITIES

1. In the Phone List program, change the filename code to collect filenames using the InputBox. Be sure to error trap with **On Error GoTo** *line*.
2. Write an error handler for frmData.cmdOK that displays an error message when the names and phone number are not entered.
3. Assume the array *TrappableErr()* is an array of strings containing the error messages of Visual Basic. The first error message could be displayed with the following code:

   ```
   MsgBox TrappableErr(1)
   ```

   Write an error handler that uses **Err**, and the array described above, to display the error message when a run-time error occurs.
4. Use the Help system to look up the various **Exit xxx** statements, such as **Exit Sub** and **Exit For**. Summarize what each does.

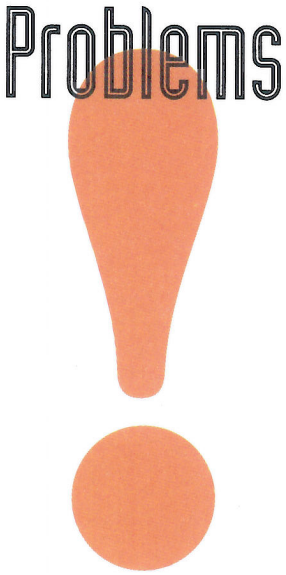
Summary

A database is a file of information. A record is the information about a particular entry. The pieces of information that make up each record are called fields. Three file types can be opened in Visual Basic: the sequential file, the random access file, and the binary file.

The **Get** and **Put** statements fetch and place records in random access files. The **EOF ()** function is **True** when a file is empty. EOF stands for "End Of File".

To keep a program from stopping when a run-time error occurs, special statements trap the error. The **On Error Resume Next** statement allows execution to proceed to the next line. **On Error GoTo** transfers program execution to a line number or line label.

A line label is a name of up to 40 characters followed by a colon. The line label provides a name for a line so that it may be referred to by a **GoTo** statement. The line label must be the first thing on the line.



1. The Random Number File Program

Write a program to generate 100 random integers in the range from 1 to 100, and put them into an array. Open a random access file named **c:\temp\rand.dat** and put the items of the array into the file. Close the file. Reopen the file, get each item, and display in a two-column List box.

2. The Computer Database Program

Write a program to collect, display, and save information about computers. Each record should contain the following information:

name of a computer	20 characters
name of processor	10 characters
speed in MHz of processor	integer
megabytes of RAM	integer
megabytes of disk space	integer
cost	currency

The internal array used to store the information should have a capacity of 25. Save the information in a file named **c:\vb3\comptrs.dat**. Record the data for each entry either through textboxes on the form or through **InputBox** statements. Display the data in textboxes on the form.

3. The School Assignment Program

Write a program to save and display a database of school assignments and scores. The record for each assignment should include the date of the assignment, a brief description, and a whole number score.

4. The Baseball Card Collection Program

Write a program to collect, save, and display information about baseball cards. Each record should contain information like:

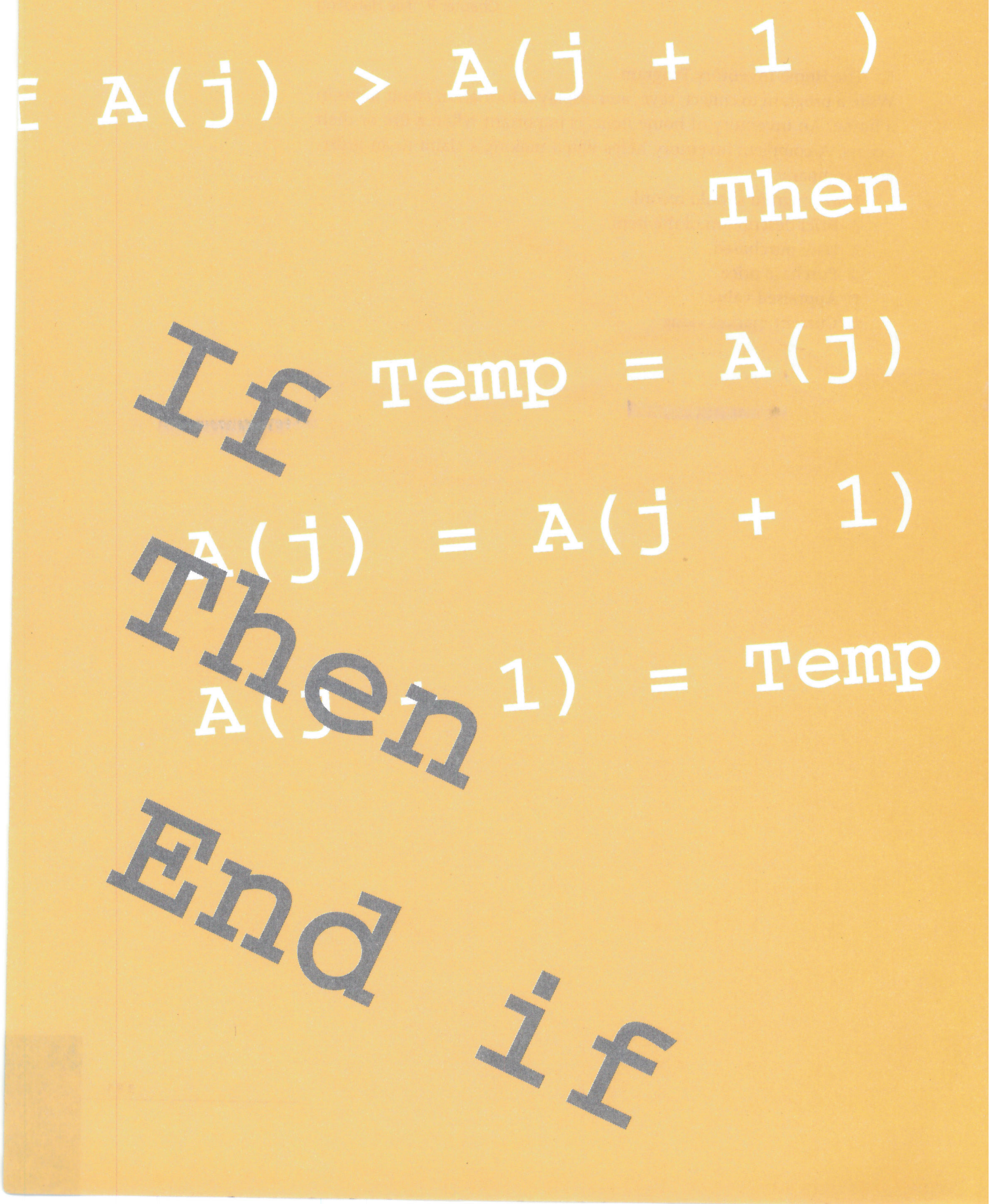
- Player's last name
- Player's first name
- Year of the card
- Team played on that year
- Value of the card
- Location of the card

Home Inventory Program

Write a program to collect, save, and display information about items in your home. An inventory of home items is important when a fire or theft occurs, making a claim to an insurance company.

Your program should record

- Brief description of the item
- Date purchased
- Purchase price
- Appraised value
- Current market value


E A(j) > A(j + 1)
Then
If
Temp = A(j)
A(j) = A(j + 1)
Then
A(1) = Temp
End if