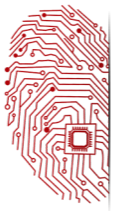


Topic: 4.3.2 File processing

In Visual Basic, **records** are known as **structures.**

A **record** is a value that contains other values, indexed by names and a **Field** is an element of a record.

Records are collections of data items (fields) stored about some object. They allow you to combine several data items (or fields) into one variable. For example, at your institute they will have a database storing a record for each student. This student record would contain fields such as Student ID, Name, and Date of Birth. The following example that DOESN'T USE RECORDS might be adequate for entering the details for one student:

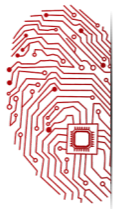
```
Dim studentID As Integer
Dim studentName As String
Dim studentDoB As Date
```

```
Sub Main()
        Console.write("insert the id: ")
        newStudentid = console.readline()
        console.write("insert the name: ")
        newStudentname = console.readline()
        console.write("insert the Date of Birth: ")
        newStudentDoB = console.readline()

        console.writeline("new record created: " &
newStudentid & " " & newStudentname & " " &
newStudentDoB)
End Sub
```

OUTPUT:

```
Code Output
insert the id: 12
insert the name: Nigel
insert the Date of Birth: 12/12/1994
new record created: 12 Nigel 12/12/1994
```



Topic: 4.3.2 File processing

But what if your institute has more than one student? We would then have to write:

```
Dim studentID1 As Integer 'field
Dim studentName1 As String 'field
Dim studentDoB1 As Date 'field
Dim studentID2 As Integer 'field
Dim studentName2 As String 'field
Dim studentDoB2 As Date 'field
Dim studentID3 As Integer 'field
Dim studentName3 As String 'field
Dim studentDoB3 As Date 'field
...
...
Dim studentID2400 As Integer 'field
Dim studentName400 As String 'field
Dim studentDoB400 As Date 'field
```

It would take an awfully long time to declare them all, let alone writing data to them. So how do we solve this? We need to combine 2 things we have learnt so far, the record and the array. We are going to make an array of student records:

```
    Structure student 'record declaration
        Dim id As Integer 'field
        Dim name As String 'field
        Dim DoB As Date 'field
    End Structure

    Sub Main()
        Dim newStudents(400) As student 'declare an
array of student records,  a school with 401 students

        for x = 0 to 400 'insert the details for each
student
            Console.WriteLine("insert the id")
            newStudents(x).id = Console.ReadLine()
            Console.WriteLine("insert the name")
            newStudents(x).name = Console.ReadLine()
            Console.WriteLine("insert the Date of
Birth")
            newStudents(x).DoB = Console.ReadLine()
        next
        for x = 0 to 400 'print out each student
            Console.WriteLine("new record created: " &
newStudents(x).id & " " & newStudents(x).name & " " &
newStudents(x).DoB)
        next
    End Sub
```

Topic: 4.3.2 File processing

In the code above, we took the concept of loops and arrays and we simply joined them to accommodate adding records one after the other and also displaying the records that have been created.

The code above allows us to create records, but we would also want to write the records to a file and then read the file to access the records later.

Opening and Closing files

The first command to include in a program which needs to work with files is the **Open** command. Open assigns the file to a numbered **file handle**, also called a **channel**, or sometimes a **buffer**. The format of the command is:

Open "Filename" [For Mode] [AccessRestriction] [LockType] As #FileNumber

For example:

Open "MyFile.txt" For Random Read Lock Read as #1

- **MyFile.txt** is the name of the file in the disk directory
- **For Random** means that access to the records can be random; if access is not specified then For random is the default value.
- **Read** restricts access to Read-only – the user cannot write or change the records.
- **Lock Read** means that only the person reading the record can have access to it at any given time; it is not shared among users.
- **As #1** means the file is assigned file handle #1; for all processing in the program, it will always be referred to as #1, and not its Filename.

AccessRestriction and **LockType** are parameters that are used mostly with files in a network environment. You use them when you want the file to be shared or not, and you want to prevent certain users from changing or deleting things that they shouldn't.

Topic: 4.3.2 File processing

For Mode in the Open statement indicated how the file will be used. There are 5 access modes:

- **Input:** open for sequential input; the file will be read sequentially starting from the beginning.
- **Output:** open for sequential output; records will be written sequentially starting at the beginning; if the file does not exist, it is created; if it does exist, it is overwritten.
- **Random:** open for random read and write; any specific record can be accessed.
- **Append:** sequential output to the end of an existing file; if the file does not exist, it is created; it does not overwrite the file.
- **Binary:** open for binary read and write; access is at byte level.

As mentioned earlier, if access mode is not specified in the Open statement, **then For Random is used by default.**

Once processing is finished, you need to **close** all the files that you have opened. The format for the **close statement** is:

Close #FileNumber1 [, #FileNumber2] …

You can close any number of files with one close statement. Eg:

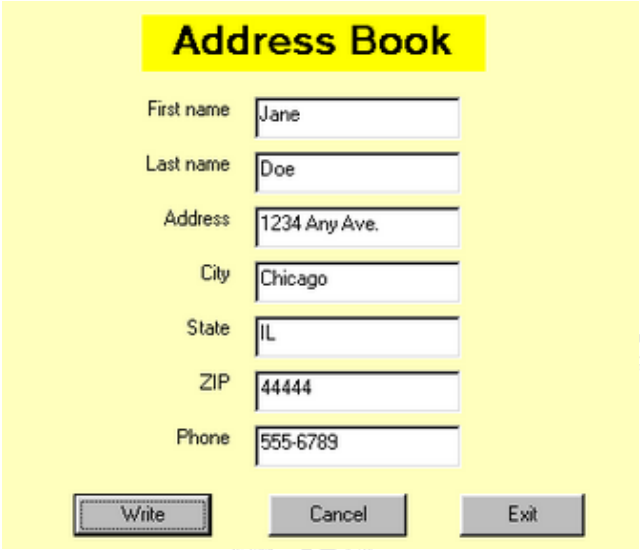
Close #1, #2, #3

If you only write "**Close**" then you can close all the open files.

Topic: 4.3.2 File processing

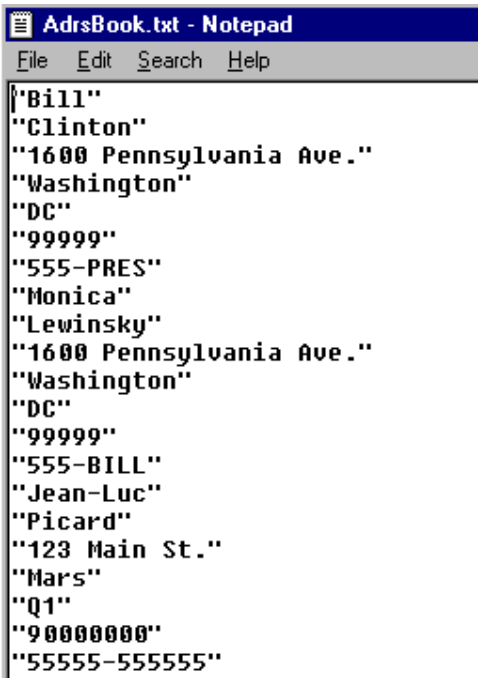
Now suppose you have an Address Book which will store the details of people such as First Name, Last Name, City, etc…

The form may look something like this:



Usually for sequential file access, records are written to a text file. So the name and location of the text file should be specified in the code.

Once the file has been created, we can use Notepad to look at it. The entry in the form above has not been written. It gets written only when you hit the Write button. Each field is stored as a separate line in the file. When we read them, we read in the same order as that in which they were written.



Topic: 4.3.2 File processing

The code to write to file is fairly straightforward. Once information has been entered into 7 TextBoxes, we use a FOR … NEXT loop to execute the Write command. The reason for this is that the Write command outputs only one field at a time. So, we have to do 7 writes to output the whole record. After the TextBoxes have been written-out, we clear then to create the next record.

```
Private Sub cb_write_Click()
    Dim intCnt As Integer
    Dim intCnt2 As Integer

    For intCnt = 0 To 6
        Write #1, txt_field(intCnt).Text
    Next intCnt

    For intCnt2 = 0 To 6
        txt_field(intCnt2).Text = ""
    Next intCnt2
End Sub

Private Sub cb_cancel_Click()
    Dim intCnt As Integer
    For intCnt = 0 To 6
        txt_field(intCnt).Text = ""
    Next intCnt

    txt_field(0).SetFocus
End Sub

Private Sub cb_Exit_Click()
    Close
    Unload Me
End Sub
```

For random access, the command to write records is **Put.** Its format is:

Put #Filenumber, [RecordNumber], Variable

RecordNumber is optional and if omitted, variable is written in Next record position after Last Put or Get statement.

The command to read records from a Random file is : **Get.** Its format is:

Get #FileNumber, [RecordNumber], Variable

If RecordNumber is omitted, next record is read from the file.

Topic: 4.3.2 File processing

Adding data

Serial file

Adding data is simple – it is added to the end of the file:

```
OPEN File in WRITE MODE
GOTO End of File
WRITE NewData
CLOSE File
```

Sequential file

The addition of data to a sequential file is more complicated than in a serial file, because the record must be inserted into the correct position – not just at the end of the file.

In practice, when a record is added to a sequential file, it is usual to create a new file and copy all the records from the old file, but insert the new record in its correct position.

An algorithm for this is shown below:

```
OPEN a NewFile in WRITE MODE
OPEN ExistingFile in READ MODE
READ First Record in ExistingFile
REPEAT
      IF key of SelectedRecord in ExistingFile < key of NewRecord THEN
            COPY SelectedRecord into NewFile
      ELSE
            COPY NewRecord into NewFile
            COPY SelectedRecord into new file
      END IF
      READ Next Record in ExistingFile
END REPEAT when new record has been copied
COPY ALL remaining records from ExistingFile into NewFile
CLOSE NewFile and ExistingFile
```

Topic: 4.3.2 File processing

Random file

Random Access Files – Key Concepts

- Used to access files faster as compared to sequential access files
- Individual records of a random-access file can be accessed directly without searching other records
- Records are accessed by their record number.

```
Type emprecord
    name1 As String * 10
    age As Integer
End Type

Dim newemp As emprecord
Open "c:\test.dat" For Random As #1 Len = Len(newemp)

Dim recno As Integer
Dim temprec As emprecord
recno = Val(Trim(Text3.Text))
temprec.name1 = Trim(Text1.Text)
temprec.age = Val(Text2.Text)
Put #1, recno, temprec
```

Topic: 4.3.2 File processing

Searching for/retrieving data

Serial file

To retrieve data from a serial file, a program must examine the first record and then all subsequent records until the desired one is found or until the end of the file has been reached.

The following algorithm does this:

```
OPEN File in READ MODE
READ First Record
SET Variable Found = False
REPEAT
      IF RequiredRecord = SelectedRecord THEN
            SET Variable Found = True
      ELSE
            READ Next Record
      END IF
END REPEAT when Found = True OR when EOF is reached
CLOSE File
```

Note that to be sure that a record does not exist in a serial file, every single record must be examined.

Topic: 4.3.2 File processing

Sequential file

Searching a sequential file is the same as searching a serial file, except that the search only has to continue until a record with a higher Key field is reached – this would mean that the data is not in the file.

```
OPEN File in READ MODE
READ First Record
SET Variables Found = False, Exists = True
REPEAT
      IF RequiredRecord = SelectedRecord THEN
            SET Variable Found = True
      ELSE
            READ Next Record
            IF Key of RequiredRecord > Key of SelectedRecord THEN
                  Exists = False
            END IF
      END IF
END REPEAT when Found = True OR Exists = False OR when EOF is reached
CLOSE File
```

Random file

```
Dim temprec As emprecord
Get #1, Val(Text3.Text), temprec
Text1.Text = temprec.name1
Text2.Text = temprec.age
```

Topic: 4.3.2 File processing

Deleting data

Serial or sequential file

```
OPEN a NewFile in WRITE MODE
OPEN ExistingFile in READ MODE
READ First Record in ExistingFile
REPEAT
      IF Key of SelectedRecord <> Key of RecordToDelete THEN
            COPY SelectedRecord into NewFile
      ELSE
            DO NOT COPY SelectedRecord
      END IF
      READ Next Record in ExistingFile
END REPEAT when EOF is reached
CLOSE NewFile and ExistingFile
DELETE ExistingFile
RENAME NewFile to Name of EixtingFile
```

Topic: 4.3.2 File processing

Random file

Unfortunately, VB6 Random Access files provide no direct mechanism for record deletion, primarily because deletion leads to a rat's nest of other issues, such as file contraction (filling the empty space), fragmentation (unused empty space), to name a couple. If you truly need to delete a record, about the only option you have is to copy all the other records to a temporary file, delete the old file, and rename the temp file to the "original" name - and, sadly, that's right from Microsoft. This is exactly what we are doing in Sequential and Serial files above.

One thing you can do, which I'll admit up front isn't ideal, is to add a "deleted" field to your random-access file, defaulting to 0, but changing to true, 1, or some other relevant value, to indicate that the record is no longer valid.

Code Example:

```
Type SampleRecord
   UserID As Long
   lastName As String * 25
   firstName As String * 25
   Deleted As Boolean
End Type
' This logically deletes a record by setting
' its "Deleted" member to True
Sub DeleteRecord(recordId As Long)

   Dim targetRecord As SampleRecord
   Dim fileNumber As Integer

   fileNumber = FreeFile

   Open "SampleFile" For Random As fileNumber Len = LenB(SampleRecord)

   Get fileNumber, recordId, targetRecord

   targetRecord.Deleted = True

   Put #fileNumber, recordId, targetRecord

   Close #fileNumber

End Sub
```

Topic: 4.3.2 File processing

```
Sub AddRecord(lastName As String, firstName As String)

   Dim newRecord As SampleRecord
   Dim fileNumber As Integer
   Dim newRecordPosition As Long


   newRecord.firstName = firstName
   newRecord.lastName = lastName
   newRecord.Deleted = False
   newRecord.UserID = 123 ' assume an algorithm for assigning this value

   fileNumber = FreeFile
   Open "SampleFile" For Random As fileNumber Len = LenB(SampleRecord)
   newRecordPosition = LOF(fileNumber) / LenB(SampleRecord) + 1
   Put #fileNumber, newRecordPosition, newRecord
   Close #fileNumber

End Sub
```