File

Finora i dati li abbiamo conservati in MC che, essendo volatile, non è proprio un buon posto dove conservare “permanentemente”. Abbiamo la soluzione alternativa, che utilizzaimo inconsapevolmente da quando usiamo un pc, ovvero registrare su MM (disco, penna usb, cd, etc) i nostri dati. Sappiamo, sempre inconsapevolmente, che i “dati su MM li possiamo vedere perchè hanno un nome e, spesso, un’icona associata ed hanno un posto (cartella) in cui sono contenuti/catalogati, a differenza dei dati in MC che non vediamo e non sappiamo nemmeno come ritrovare.

I dati che normalmente conserviamo sono: un documento Word, un foglio Excel, un disegno con Paint, etc.... Una volta creato, lo salviamo dandogli un nome (scelto liberamente da noi) e (non scelto da noi) una estensione che indica il sw che lo ha creato (e che se manca non permette al S.O. di sapere con che sw “considerare” il file. Possiamo organizzare i nostri file creando cartelle (con sottocartelle e ulteriori sottocartelle, e.... ) creando così un ALBERO gerarchico di cartelle la cui radice è il l’unità a disco che viene detta C: (se è una penna usb potrebbe avere un nome tipo “pippo”, “optima”, etc...). Attualmente tutti i S.O. permettono di organizzare il FILE SYSTEM ad albero.

Ma non solo i vari sw del pacchetto Office (e gli altri che troviamo di base in W. Quale Blocco Note, Paint, etc...) permettono di registrare permanentemente (ovvero domani, dopodomani, fra un anno... ritroverò il file) ma anche i linguaggi di programmazione hanno istruzioni per creare, modificare, cancellare file che contengano dati più o meno organizzati.

Questo significa che tutti gli esercizi che finora abbiamo scritto e che prevedevano OGNI VOLTA, l’immissione dei dati, potrebbero diventare con dati che, una volta registrati, posso ritrovare anche successivamente. Questa caratteristica è alla base del sw gestionale!

## **I file ad accesso sequenziale**

Un file, di qualsiasi tipo esso sia, per poter essere utilizzato deve necessariamente essere **aperto** (poichè è un “posto” dove “conservare” dei dati... come un armadio... devo aprirlo per metterci le cose e... chiuderlo quando ho finito!). L’apertura si esegue con l’istruzione **FileOpen**, la cui sintassi, per le strutture ad accesso sequenziale, è la seguente:

**FileOpen (numeroFile, percorso\file, modalità)** dove:

* numeroFile è un numero che identificherà il file univocamente all'interno del programma. Tale numero corrisponde al canale (buffer) attraverso cui transiteranno le informazioni tra la memoria centrale e il file e sarà utilizzato come nome interno in tutte le successive operazioni di lettura, scrittura e chiusura del file. Deve essere un intero compreso fra 1 e 511. Corrisponde ad un’area in MC che fa da posto di smistamento per i dati da e verso le MM
* percorso\file è una stringa (*esplicita*, costante o variabile) che indica il percorso (facoltativo) e il nome (obbligatorio) del file che si desidera aprire. E' detto anche nome esterno e va specificato solo al momento dell'apertura.
* modalità indica l'intento con cui si accede al file, *che non può mutare fino alla chiusura del file stesso*. Le possibili modalità sono:
  + **OpenMode.Output** crea un nuovo file vuoto su disco, predisponendolo per la scrittura delle informazioni. Se il file è già presente sul disco, viene prima cancellato.
  + **OpenMode.Append** apre un file esistente su disco, al fine di accodare altri record a quelli già presenti. Se il file non viene trovato, ne viene creato uno nuovo senza alcun messaggio d'errore e il funzionamento coincide in pratica col caso precedente.
  + **OpenMode.Input** il file è aperto in lettura, ovvero è utilizzato esclusivamente per reperire informazioni senza poter effettuare su di esse alcuna modifica. È ovvio che l'apertura di un file in questa modalità richiede che esso sia stato in precedenza creato sul disco, altrimenti provoca un errore che il File System notifica al programma in esecuzione.

Esempi:

FileOpen (1, "C:\Personale\Dipendenti.seq", OpenMode.Output)

FileOpen (2, "\Personale\Clienti.seq", OpenMode.Append)

FileOpen (3, ".\Prodotti.seq", OpenMode.Input) FileOpen (4, "Magazzino.seq", OpenMode.Append)

oppure

Const FileO = "C:\Personale\Dipendenti.seq"   
FileOpen (1, FileO, OpenMode.Output)

Const FileI = ".\Prodotti.seq"   
FileOpen (3, FileI, OpenMode.Input)

Per scrivere dati in un file sequenziale, si apre il file in modalità OpenMode.Output o OpenMode.Append, si usa l’istruzione:

**PrintLine (numeroFile, dato)** dove numeroFile è il numero che identifica il file e dato è il dato da scrivere nel file.

Esempio: PrintLine (1, Cognome)   
 PrintLine (1, Nome)

Ogni istruzione PrintLine esegue una scrittura di un dato. I dati sono scritti in sequenza con un “a capo” dopo ogni scrittura e possono essere anche di tipi diversi.

Per leggere dati da un file sequenziale, si apre il file in modalità OpenMode.Input e si usa l’istruzione:

**dato = LineInput(numeroFile)** dove numeroFile è il numero che identifica il file e dato è il dato da leggere dal file. I dati vengono letti nell’ordine esatto con cui sono stati scritti.

Esempio: Cognome= LineInput (1) Nome = LineInput (1)

Quando un file viene aperto in lettura, il puntatore di lettura è posizionato all'inizio del primo dato registrato. Ogni lettura di un dato fa avanzare il puntatore all'inizio del dato successivo. Per sapere se la lettura ha raggiunto la fine del file, è necessario ricorrere alla funzione EOF (End Of File), alla quale, come di consueto, va passato il numero del file come parametro. Il valore restituito è di tipo logico e vale True se è stata raggiunta la fine del file, False in caso contrario. Si noti che nei file sequenziali EOF diventa True non appena viene letto l'ultimo dato.   
Per scorrere tutto il contenuto di un file sequenziale, è allora necessario utilizzare un ciclo del tipo:

**Do While** Not EOF(n)

Dato=LineInput (n)

<trattamento del dato letto>

**Loop**

Prima di aprire un file in lettura, il programma dovrebbe accertarsi che il file esista sul disco altrimenti il programma va in errore. La funzione **Dir(percorso\file)** restituisce una stringa vuota se non trova il file. Esempio:

**Private Sub** ...

Const FileI = "Prodotti.seq" 'File di Input ...

**If** Dir(FileI) = "" **Then**

Msgbox "File inesistente: " & FileI, vbCritical, "Attenzione!"

**Else**

FileOpen (1, FileI, OpenMode.Input)

<lettura e trattamento del file>

**End If**

**End Sub**

Quando le operazioni sul file sono terminate, il file deve essere chiuso. Questa operazione svuota il buffer, portando a termine le eventuali scritture in sospeso. Inoltre rilascia il numero di buffer, che diventa riutilizzabile per la gestione di un altro file (pratica sconsigliata per ragioni di chiarezza: meglio attribuire a ogni nome esterno sempre lo stesso numero interno). Ciò comunque non pregiudica la successiva riapertura dello stesso file nel medesimo programma, con la stessa o una differente modalità.

La sintassi è:

**FileClose(numeroFile)** per chiudere il file di cui è indicato il numero

oppure semplicemente: **FileClose()** per chiudere tutti i file aperti.

Esempi: FileClose(1) FileClose(3) FileClose()

Un file sequenziale non permette che i record vengano sovrascritti per un’eventuale correzione (detta anche modifica), né che vengano cancellati, né inseriti, se non dopo l’ultimo (append). Per raggiungere ugualmente tali obiettivi in un file sequenziale è necessario:

- aprire in lettura il file da modificare;   
- aprire in scrittura (output) un file d’appoggio;   
- copiare tutti i record dal primo file al secondo, avendo cura di effettuare le correzioni necessarie prima di scrivere nel secondo file;   
- chiudere i due file;   
- cancellare il primo file con Kill (percorso\file);   
- rinominare il secondo file con il nome che aveva il primo, con Name (vecchio\_nome As nuovo\_nome);

Un file sequenziale può essere usato per memorizzare anche un testo. In questo caso ogni operazione di lettura/scrittura potrà riguardare, ad esempio, una riga del testo.

## **File ad accesso diretto (random)**

I file sequenziali presentano un limite: la scarsa idoneità all'utilizzo di grandi quantità di dati, a causa degli elevati tempi di ricerca. I file ad accesso diretto (o casuale, o random) sono invece dotati di una struttura rigida che li rende in grado di offrire maggiore velocità di accesso alle informazioni.

I blocchi d’informazioni (record) registrati in un file random devono avere tutti la stessa struttura e la stessa lunghezza.   
La struttura di un record è definita, come nuovo tipo di dati, attraverso la frase

**Structure: Public Structure nome\_tipo   
Dim nomecampo\_1 As tipo\_1   
Dim nomecampo\_2 As tipo\_2 …   
Dim nomecampo\_n As tipo\_n   
End Structure**

All’interno della struttura devono essere dichiarati gli elementi che la compongono, denominati campi. Per ognuno di essi deve essere specificato il nome e il tipo, che può essere di tipo semplice (Byte, Integer, Long, Single, Double, Boolean, stringa a lunghezza fissa) o strutturato (record o array). Un tipo così definito è denominato tipo record e deve essere dichiarato nell’area generale della classe form o in un modulo. Se un campo è di tipo stringa, deve essere preceduto dalla specifica della sua lunghezza massima, visto che i record devono avere tutti la stessa lunghezza:   
**<VBFixedString(n)>** Dove n è la lunghezza massima del campo.

Esempio:

**Structure T\_Automobile   
<VBFixedString(15)> Dim Marca As String   
<VBFixedString(30)> Dim Modello As String   
Dim Cilindrata As Integer   
Dim Diesel As Boolean   
End Structure**   
In questo modo si è provveduto ad aggiungere un nuovo tipo di dati a quelli standard previsti da Visual Basic. È quindi possibile dichiarare, ad esempio, la variabile Automobile di tipo T\_Automobile digitando:

**Dim Automobile As T\_Automobile**

Per assegnare valori ad una variabile di tipo record, si esegue un’operazione di assegnazione per ogni campo, secondo la sintassi: variabile.campo = valore

Ad esempio, volendo assegnare il valore 1700 al campo Cilindrata della variabile Automobile, definita in precedenza, occorre scrivere:

Automobile.Cilindrata = 1700

Come per i file sequenziali, l’apertura di un file ad accesso diretto avviene per mezzo dell’istruzione FileOpen che, in questo caso, assume la forma:

**FileOpen (numeroFile, percorso\file, modalità, , , lunghezzaRecord)**

La modalità di accesso è **OpenMode.Random** qualunque sia l’operazione che si desidera effettuare sul file. Al parametro lunghezzaRecord deve essere assegnata l’esatta dimensione del record in byte, che può essere determinata per mezzo della funzione **Len**, il cui parametro è una qualunque variabile del tipo record del file. Ad esempio:

FileOpen (1, “Concessionaria”, OpenMode.Random, , , Len(Automobile))

La presenza delle virgole prima del parametro di lunghezza del record è motivato dal fatto che l’istruzione File Open completa prevede altri due parametri facoltativi prima di specificare la lunghezza del record.

Anche i file ad accesso casuale devono essere chiusi dopo l’uso per mezzo dell’istruzione FileClose.

Per scrivere il contenuto **di un record** in un file ad accesso casuale si usa l’istruzione:   
FilePut (numeroFile, variabile, posizione)

dove numeroFile è il numero del buffer con cui è stato aperto il file, variabile è il nome della variabile contenente i dati da scrivere e il parametro posizione indica il numero della posizione in cui scrivere. I record sono numerati a partire da 1.

Esempio:

FilePut (1, R, 5)

scrive il contenuto della variabile di tipo record R nella quinta posizione del file con buffer 1. Se il valore del parametro posizione individua un record già presente in archivio, esso viene sovrascritto. Per aggiungere un nuovo record al file (operazione equivalente all’append dei file sequenziali) è invece necessario indicare il numero del record successivo all’ultimo. Nei file random è possibile calcolare facilmente il numero dei records presenti nel file (perché hanno tutti la stessa lunghezza) mediante la formula:

**numeroDeiRecordsNelFile = lunghezzaDelFileInByte / lunghezzaRecordInByte**

La lunghezza del file può essere calcolata usando la funzione LOF(n), dove n è il numero del buffer. La lunghezza di un record può invece essere calcolata attraverso la funzione LEN(v), dove v è una variabile del tipo record dichiarato per il file.

Il numero del record **successivo** all’ultimo si ottiene aggiungendo 1 al numero complessivo dei records:

**nr = LOF(n)\LEN(V)+1**

La lettura di un record contenuto in un file ad accesso casuale avviene per mezzo dell’istruzione: FileGet (numeroFile, variabile, posizione)

dove numeroFile è il numero del buffer con cui è stato aperto il file, variabile è il nome della variabile in cui i dati letti devono essere posti e il parametro posizione indica il numero del record da leggere, compreso fra 1 e il numero dei records presenti nel file. Ad esempio, per leggere il quarto record del file di buffer 1 e porre il suo contenuto nella variabile R, l’istruzione è:

**FileGet (1, R, 4)**

Solo in un caso il valore del parametro posizione può essere omesso: quando le operazioni di lettura avvengono in sequenza; l’assenza del numero indicante la posizione provoca infatti l’accesso al record successivo a quello corrente Ad esempio, la sequenza: FileGet (1, R, 4) FileGet (1, V) provoca la lettura, nelle variabili R e V, del quarto e del quinto record del file con buffer 1.

Il metodo più semplice per cancellare un record consiste nel sovrascriverlo con un elemento vuoto. In questo modo, tuttavia, non è possibile recuperare lo spazio da esso occupato sul disco, si tratta cioè di una cancellazione logica, non fisica. La cancellazione logica dei record offre però un’importante opportunità: fare in modo che elementi eliminati per errore possano essere agevolmente recuperati. Ciò è possibile aggiungendo un campo booleano alla struttura dei record e facendo in modo che il programma che accede all’archivio consideri cancellati tutti gli elementi contenenti il valore logico True all’interno di questo campo. L’eliminazione di un record comporta quindi la semplice variazione del valore di un suo campo. Analogamente, è possibile recuperare un elemento cancellato per errore impostando nuovamente al valore False il campo booleano. La struttura T\_Automobile può a tal fine essere modificata come segue:

Public Structure T\_Automobile   
<VBFixedString(20)> Dim Marca As String   
<VBFixedString(50)> Dim Modello As String Dim   
Cilindrata As Integer   
Dim Diesel As Boolean   
Dim Cancellato As Boolean   
End Structure

Quando la quantità di informazioni da gestire diventa elevata, la necessità di recuperare lo spazio occupato dai record cancellati diventa evidente, sia per liberare spazio su disco, sia per ridurre i tempi di accesso alle informazioni, che aumentano costringendo il programma a leggere dati inutili. Basta allora realizzare una procedura in grado di creare un nuovo file e copiare al suo interno tutti i record non vuoti. Successivamente, si deve eliminare il vecchio file ed assegnare il suo nome alla nuova struttura.

Scorrere il contenuto di un file random implica la necessità di sapere quando si arriva alla fine del file. Come per i file sequenziali, è disponibile la funzione EOF, per cui il ciclo di lettura può, ad esempio, essere il seguente:

**Do While Not EOF(1)  
FileGet (1, Automobile)** ‘ elaborazione dati record letto … **Loop**

Potendo però calcolare il numero dei record nel file, si può anche usare un semplice ciclo For…Next:   
**NR=LOF(1)/LEN(Automobile)   
For I = 1 to NR   
FileGet (1, Automobile, I)** ‘ elaborazione dati record letto …   
**Next**