

[Home Page](#) [Novità](#) [Aiuto](#) 

Corso Intermedio - Lezione 7

http://www.vbsimple.net/intermed/int_07.htm

- [L'istruzione Get per la lettura di dati da file.](#)
 - [Le funzioni Space e String.](#)
 - [Effettuare il ciclo di lettura.](#)
 - [La funzione EOF.](#)
 - [Inserimento di righe in un controllo ListBox.](#)
-

Fino alla lezione precedente ci siamo preoccupati di scrivere i dati dei brani caricati, all'interno del file di dati mediante l'istruzione **Put**. L'operazione si completa con l'istruzione dedicata alla lettura di dati da un file binario: **Get**. Il suo funzionamento è pressoché uguale a quello dell'istruzione Put la cui sintassi è qui descritta:

```
Get [#]<numerofile>, [<posizione>], <buffer>
```

Il primo dei tre argomenti è il numero del file già aperto mediante istruzione Open e può - ma non necessariamente - essere preceduto dal simbolo #; l'argomento è comunque obbligatorio e determina il file da cui verranno letti i dati. Il secondo argomento è opzionale e indica la posizione da cui leggere i dati, espressa in numero del record all'interno del file ad accesso Random, oppure in numero di bytes per files ad accesso Binary. Se il secondo argomento non viene specificato, sarà letto il record successivo a quello dell'ultima operazione di lettura dal file.

Il terzo argomento è obbligatorio e rappresenta il **buffer** in cui saranno memorizzati i dati provenienti dal file. Tale area dovrà essere precedentemente preparata, in quanto la quantità di dati letti dal file dipenderà dall'ampiezza della suddetta variabile. Ciò significa che per effettuare la lettura dovrà essere utilizzata una variabile dello stesso tipo contenuto nel file. Ad esempio se durante la scrittura dei dati è stata utilizzata una variabile di tipo *Integer*, per effettuare una corretta lettura dovrà essere utilizzata una variabile dello stesso tipo.

Particolare cura deve essere dedicata nella lettura di stringhe da un file ad accesso Random o Binary. Come detto, la variabile che ospiterà i dati letti dal file dovrà essere dello stesso tipo scritto all'interno del file; a ciò si aggiunge la necessità di specificare, prima della lettura, l'ampiezza della stringa da leggere.

Le *stringhe a lunghezza fissa*, come spiegato nella [lezione 4](#), mantengono comunque l'ampiezza specificata durante la progettazione, indipendentemente dai dati in esse contenuti, riempiendo con spazi a destra o troncando la stringa ove necessario. Ciò comporta una scrittura ed una lettura sempre uguale dal file poiché la dimensione del buffer è comunque costante.

Utilizzando invece *stringhe a lunghezza variabile*, ovvero stringhe senza una lunghezza

dichiarata è richiesta l'allocazione del buffer prima dell'operazione di lettura; ciò può essere fatto semplicemente assegnando alla variabile buffer una stringa lunga tanti caratteri quanti sono i bytes da leggere dal file; ad esempio assegnando la stringa "abc" ad una variabile buffer utilizzata dall'istruzione Get determina il risultato dell'operazione di lettura; saranno infatti letti 3 bytes dal file o anche meno nel caso ci trovassimo alla fine del file, mentre la parte eccedente del buffer sarà riempita con caratteri dal valore [ASCII 0](#).

Per preparare le variabili da utilizzare quali buffer di lettura di files binari due funzioni si rivelano due ottimi alleati per semplicità d'uso: si tratta delle funzioni **Space** e **String**. Entrambe restituiscono in uscita una stringa composta da tanti caratteri quanti ne sono specificati nel primo argomento; la definizione delle due funzioni è mostrata di seguito:

```
Function Space(Number As Long)
Function String(Number As Long, Character)
```

La prima delle due funzioni, **Space**, richiede un solo argomento: il numero di caratteri che comporranno la stringa restituita. Essa sarà infatti costituita di tanti spazi quanti specificati nel richiamo della funzione. Nessun altro carattere sarà presente nella stringa restituita.

La seconda funzione, **String**, da non confondere con il tipo di dati, richiede invece due argomenti: il primo specifica il numero di caratteri che comporranno la stringa mentre il secondo specifica il carattere di cui sarà costituita la stringa; tale argomento può essere un numero, corrispondente al codice ASCII del carattere da replicare altrimenti può essere una stringa il cui primo carattere verrà replicato tante volte. Entrambe le chiamate alla funzione sono valide e restituiscono tutte il medesimo risultato:

```
MsgBox String(10, 65)
MsgBox String(10, Asc("A"))
MsgBox String(10, "A")
```

Aggiungiamo anche la precisazione che di entrambe le funzioni esistono due versioni: la prima è quella appena accennata, che restituisce in uscita un valore [Variant](#) di sottotipo String, quindi un tipo di dati con un'ampia occupazione; la seconda versione delle funzioni è quella che restituisce valori String senza utilizzare Variant e le funzioni prendono il nome di **Space\$** e **String\$**.

Tornando al nostro progetto, ci occuperemo di rileggere i dati salvati nella lezione precedente avendo utilizzato l'istruzione Put. L'operazione utilizzerà semplicemente l'istruzione Get vista in precedenza. Sarà specificato come buffer di lettura lo stesso record a dimensione fissa utilizzato per la scrittura; nell'esempio precedente era dato dalla variabile **udtDiscoCorrente**. Pertanto, subito dopo aver effettuato l'apertura del file utilizzeremo una riga simile alla seguente:

```
Get #intFileDati, , udtDiscoCorrente
```

Così facendo leggeremo un record dal file aperto e identificato dal numero **intFileDati** e lo inseriremo nella variabile **udtDiscoCorrente**.

Quest'operazione tuttavia la lettura di un singolo record dall'archivio. Il problema rimanente è quello di determinare quante righe è necessario leggere dal file e quindi

ripetere l'istruzione tante volte. Potremmo sfruttare la funzione LOF (Length Of File) trattata nella lezione precedente e stabilire un ciclo For..Next per il numero di elementi contenuti nel file dato dalla semplice divisione della lunghezza del file per la lunghezza della singola riga:

```
Elementi = LOF(intFileDati) / Len(udtDiscoCorrente)
```

Quest'operazione tuttavia richiede l'uso di due variabili: la prima che contenga il numero di elementi presenti nel file e la seconda da utilizzare come contatore nel ciclo di lettura, come segue:

```
Elementi = LOF(intFileDati) / Len(udtDiscoCorrente)
For intLoop = 1 To Elementi
  Get #intFileDati, , udtDiscoCorrente
  ...
Next intLoop
```

Un'aiuto efficace può fornircelo la funzione *EOF* (End Of File), la quale restituirà un valore booleano ad indicare che il punto di lettura sia arrivato alla fine del file, oltre la quale non resta nulla da leggere. La sintassi della funzione è molto semplice:

```
Function EOF(FileNumber As Integer) As Boolean
```

FileNumber naturalmente indica il numero del file già aperto con l'istruzione Open. Potremmo quindi sostituire il ciclo precedente con questo, ben più semplice:

```
Get #intFileDati, , udtDiscoCorrente
Do While Not EOF(intFileDati)
  ...
  Get #intFileDati, , udtDiscoCorrente
Loop
```

Non sarà necessario predeterminare il numero di record presenti nel file e non sarà altresì necessario stabilire un contatore per tutti gli elementi del file. Il ciclo infatti è preceduto da una prima lettura dal file, in seguito alla quale inizia il ciclo vero e proprio che durerà fintanto che il punto di lettura non raggiungerà la fine del file. Naturalmente all'interno del ciclo dovrà essere presente un'ennesima istruzione di lettura.

Raggiunta l'ultima riga, un'ennesima istruzione di lettura recupererà un record vuoto e sposterà il puntatore di lettura alla fine del file, cosicché la funzione *EOF* restituirà valore True ed il ciclo si interromperà.

In entrambi gli esempi tra una lettura e l'altra sono indicate con i puntini (...) le istruzioni che caricheranno il brano recuperato nell'elenco dei brani. L'operazione di inserimento di una riga in un controllo **ListBox** solitamente si compone di una sola riga che richiama il metodo  *AddItem* dell'elenco:

```
Lista.AddItem(Item As String, [Index])
```

Item specifica il testo da inserire nell'elenco ed è l'unico argomento obbligatorio; *Index*

invece specifica la posizione nella quale l'elemento sarà inserito: il valore 0 specifica il primo elemento, 1 il secondo e così via... Se l'argomento *Index* non viene specificato, l'elemento viene aggiunto in coda alla lista. È quindi possibile recuperare l'indice che l'ultimo elemento inserito ha assunto, semplicemente leggendo la proprietà *NewIndex* del controllo.

Inoltre, ad ogni elemento nella lista è possibile associare un numero non univoco per pura comodità del programmatore. Nessuna funzione utilizzerà tale numero ed ogni elemento nella lista assumerà valore 0. La proprietà che specifica tale numero è *ItemData* e richiede l'indice dell'elemento di cui recuperare o assegnare il numero. Un'operazione di inserimento può essere la seguente:

```
Lista.AddItem "ciao"
Lista.ItemData(Lista.NewIndex) = 10
```

Quest'operazione inserisce l'elemento **"ciao"** in fondo alla lista ed assegna allo stesso un valore numerico di **10**.

Vediamo quindi il codice completo del ciclo di lettura aggiornato con gli ultimi concetti trattati, evidenziati dal colore blu:

```
1. Private Sub Form_Load()
2.     With udtDiscoCorrente
3.         txtTitoloBrano.MaxLength = Len(.strTitoloBrano)
4.         txtAutoreBrano.MaxLength = Len(.strAutore)
5.         txtTitoloCD.MaxLength = Len(.strTitoloCD)
6.         txtNote.MaxLength = Len(.strNote)
7.         txtNumeroTraccia.MaxLength = 2
8.         txtAnnoPubblicazione.MaxLength = 4
9.         cmdEliminaBrano.Enabled = False
10.        cmdModificaBrano.Enabled = False
11.
12.        intFileDati = FreeFile
13.        Open App.Path & "\BRANI.DAT" For Random As intFileDati Len = Len
14.        (udtDiscoCorrente)
15.        Get #intFileDati, , udtDiscoCorrente
16.        Do While Not EOF(intFileDati)
17.            If .blnEliminato = False Then
18.                lstBrani.AddItem .strTitoloBrano
19.                lstBrani.ItemData(lstBrani.NewIndex) = .intIndice
20.            End If
21.        Get #intFileDati, , udtDiscoCorrente
22.        Loop
23.    End With
24. End Sub
```

Il valore *ItemData* che assegneremo a ciascun brano recuperato dal file corrisponde all'indice del brano, anch'esso recuperato dal file. Inizialmente potrà sembrare inutile ma la sua utilità si potrà notare quando l'elenco verrà modificato, ad esempio per ordinarlo per titolo, anziché per indice.

Il codice fin qui trattato si limita ad inserire una riga nell'elenco dei brani ed a ricaricare l'elenco all'avvio del programma. Vedremo nella prossima lezione come effettuare una lettura in maniera casuale, ovvero una lettura mirata ad un elemento specifico, recuperato dall'elenco già caricato.

[Eibia FBI](#)

16 Settembre 2003



[Torna alla sesta lezione](#)

[Vai all'ottava lezione](#)

