



## Corso Intermedio - Lezione 10

[http://www.vbsimple.fbi/intermed/int\\_10.htm](http://www.vbsimple.fbi/intermed/int_10.htm)

- [Verifica dell'input.](#)
- [L'evento KeyPress.](#)
- [Gestione degli errori.](#)

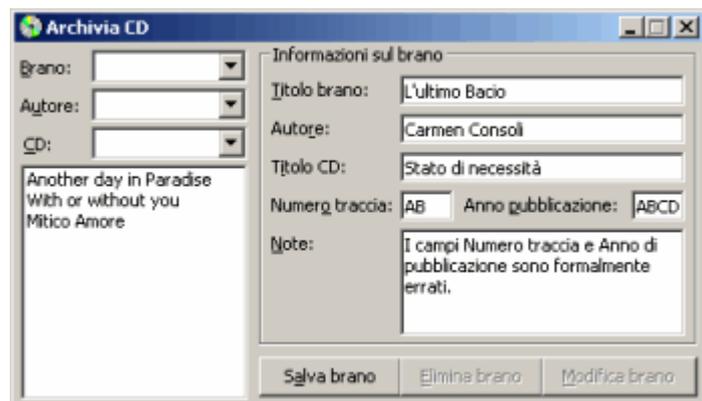
Ogni programma che consente l'immissione di dati da parte dell'utente deve verificarne la validità sia formale che di congruenza. È importante tenere a mente che tutti i dati inseriti dall'utente potrebbero essere errati, ad esempio esserci lettere anziché numeri o simboli strani anziché una data; anche il nostro progettino sin qui sviluppato necessita di queste verifiche: i campi **Numero traccia** e **Anno di pubblicazione** si aspettano dei dati numerici e nessuna verifica è effettuata né a priori né a posteriori.

Proviamo ad esempio ad inserire delle lettere nei due campi numerici, come mostrato nella **Figura 1**. Le due caselle di testo hanno ricevuto i dati anziché rifiutarli durante la digitazione; questo è già un comportamento inaccettabile e fuorviante per l'utilizzatore, ma procediamo oltre...

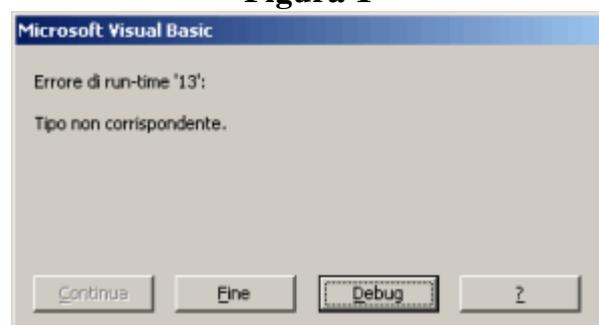
Alla pressione del pulsante **Salva brano**, nel tentativo di salvare i dati su file, il programma si interrompe con l'errore 13, mostrato nella **Figura 2**. Se il programma non fosse eseguito all'interno dell'ambiente di Visual Basic si sarebbe chiuso con l'errore mostrato nella **Figura 3**.

Si rende perciò necessario aggiungere dei controlli formali sulle caselle di Input che si aspettano l'inserimento di cifre numeriche. A tal fine impediremo l'inserimento di qualsiasi carattere che non sia effettivamente un numero.

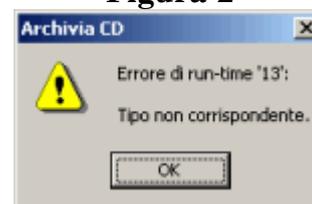
Questo genere di controlli solitamente si suddivide in due parti: la prima riguarda la digitazione e [la seconda riguarda la correttezza](#). Il primo di questi controlli si basa



**Figura 1**



**Figura 2**



**Figura 3**

sull'utilizzo dell'evento  **KeyPress**, presente in quasi tutti i controlli; vediamolo subito in dettaglio:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
```

L'evento fornisce un parametro di nome **KeyAscii** [passato per riferimento](#) (notare l'assenza della parola chiave *ByVal*) che corrisponderà al codice [ASCII](#) del tasto premuto, quindi se relativo ad un casella di testo, corrisponderà anche al carattere da inserire.

Esiste inizialmente questa corrispondenza tra codice del tasto premuto e carattere da inserire: nel momento in cui l'utente premerà il tasto **A** sarà richiamato l'evento **KeyPress** ed il parametro **KeyAscii** avrà valore 65, corrispondente al codice ASCII di A. Essendo il parametro passato per riferimento ogni modifica al suo valore apportata all'interno della routine sarà riportata anche all'esterno; ciò significa che se variassimo il valore del parametro **KeyAscii** sarà rappresentato un altro carattere. Tornando all'esempio precedente:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
    If KeyAscii = 65 Then KeyAscii = 66
End Sub
```

Il presente codice nel caso in cui il codice del tasto premuto fosse 65 (la A maiuscola), modificherà il valore di **KeyAscii** in 66 (B maiuscola); questo comporta che ogni volta che l'utente tenta di scrivere A nella casella di testo, al suo posto sarà scritto B. Alla stessa maniera è possibile richiedere che scartare il carattere e non scrivere nulla; basterà semplicemente assegnare valore 0 al parametro **KeyAscii**.

Pertanto, al fine di negare la possibilità di scrivere lettere o simboli nelle caselle in cui desideriamo accettare soltanto numeri, dovremo fare un semplice controllo del valore di **KeyAscii** e se questo non dovesse essere una cifra, scartare il carattere ricevuto.

L'operazione è molto semplice:

```
1. Private Sub txtAnnoPubblicazione_KeyPress(KeyAscii As Integer)
2.     If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And (KeyAscii <> vbKeyBack) Then
3.         KeyAscii = 0
4.     End Sub
5. Private Sub txtNumeroTraccia_KeyPress(KeyAscii As Integer)
6.     If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And (KeyAscii <> vbKeyBack) Then
7.         KeyAscii = 0
8.     End Sub
```

Per entrambe le caselle di testo **txtAnnoPubblicazione** e **txtNumeroTraccia** sarà effettuato lo stesso controllo: se il valore di **KeyAscii** non è compreso tra i codici dei tasti 0 e 9 (ovvero è minore di 0 oppure maggiore di 9) il tasto andrà scartato. A questo controllo ne è affiancato un altro: la battuta andrà scartata se il tasto premuto non è neanche quello di cancellazione (*Backspace*). Senza quest'ultimo controllo la casella avrebbe accettato soltanto i numeri, ma non sarebbe stato possibile cancellare le battute utilizzando il tasto **Backspace**.

Esistono anche altre soluzioni per rendere una casella di testo numerica e sono trattate nell'[HowTo dedicato alla TextBox numeriche](#).



Nel codice in esame sono state utilizzate le costanti `vbKey0` e `vbKey9` per indicare i codici dei tasti 0 e 9; al loro posto avremmo potuto utilizzare i valori 48 e 57, così come le chiamate a funzione `Asc("0")` e `Asc("9")`. In tale situazione è sempre bene cercare di sfruttare le costanti messe a disposizione da Visual Basic, rendendo peraltro il codice più scorrevole e leggibile.

---

Alla dimostrazione pratica possiamo notare che questo controllo è efficace ad inibire tutte le battute che non siano cifre numeriche; resta tuttavia ancora una possibilità di errore data dalla libertà all'utente di sfruttare le capacità di Incollare del testo precedentemente copiato negli [appunti](#), sfruttando il menu popup oppure la combinazione di tasti CTRL+V.

Nonostante esista la possibilità di evitare questo genere di comportamenti vogliamo mettere da parte questa difficoltà per introdurre il secondo dei controlli: quello relativo alla correttezza dei dati recuperati ed alla relativa **gestione degli errori**.

È opportuno tenere a mente che in qualsiasi operazione importante i dati da trattare potrebbero essere formalmente errati; di solito ciò è dovuto ad una cattiva analisi del problema oppure al sorgere di un comportamento inizialmente non previsto ed ancora i dati potrebbero essere stati corrotti su disco da Virus o malfunzionamenti.

È compito del buon analista programmatore cercare di prevedere tutti i possibili comportamenti dell'utente e compito del programmatore mantenere sempre il controllo del funzionamento del programma, cercando di gestire anche i comportamenti non previsti. Supponiamo che la possibilità di incollare del testo nelle caselle di testo sia un comportamento non previsto; sappiamo che in qualche maniera l'utente riuscirà a scrivere un valore non numerico in quelle caselle che si aspettano un valore numerico.

L'utilizzo di un dato non numerico al posto di uno numerico abbiamo visto che comporta un errore *13 - Tipo non corrispondente*, come mostrato nelle **Figure 2 e 3**. È però possibile gestire gli errori con l'apposita istruzione:

```
On Error <comportamento>
```

Dove al posto di <comportamento> possiamo avere una delle seguenti istruzioni:

- `Resume Next`  
Ignora l'errore e procede alla riga successiva
- `Goto 0`  
Disattiva la gestione degli errori
- `Goto <Etichetta>`  
Al verificarsi dell'errore il codice salterà alla posizione successiva all'etichetta indicata

Dopo il salto ad un'etichetta è anche possibile tornare alla riga che ha generato l'errore oppure a quella immediatamente successiva utilizzando le istruzioni `Resume` e `Resume Next` rispettivamente. Particolare cura deve essere dedicata nell'uso dell'istruzione `Resume`

perché l'esecuzione tornerà alla stessa riga che ha generato l'errore; se la situazione di errore non viene corretta si genera un meccanismo infinito di genera errore-gestisci-ritenta.

In ogni caso al verificarsi di un errore gestito o non gestito, l'oggetto globale **Err** sarà riempito con le informazioni sull'errore verificatosi; in tal modo, durante un'operazione di gestione di errori, sarà possibile cercare di capire che errore si è verificato e quindi decidere cosa fare successivamente. Vediamo alcuni esempi per capire meglio il funzionamento:

- `Open "C:\FILE_INESISTENTE" For Input As #1`

Sarà generato un errore 53 - *Impossibile trovare il file* ed il programma sarà interrotto.

- `On Error Resume Next`  
`Open "C:\FILE_INESISTENTE" For Input As #1`

Sarà generato comunque l'errore 53 ma non sarà notificato all'utente e l'esecuzione del programma procederà alla prossima istruzione; il file ovviamente non sarà aperto.

- `On Error Resume Next`  
`Open "C:\FILE_INESISTENTE" For Input As #1`  
`If Err.Number <> 0 Then Exit Sub`

Sarà generato comunque l'errore 53 ma non sarà notificato all'utente e l'esecuzione del programma procederà alla prossima istruzione, che controllerà la generazione di un errore ed in tal caso richiederà l'uscita dalla procedura.

- `On Error Goto Gestione_Errori`  
`Open "C:\FILE_INESISTENTE" For Input As #1`  
`Close #1`  
`Gestione_Errori:`  
`MsgBox "Si è verificato un errore.", vbCritical`

Sarà generato l'errore 53 ma come in precedenza, in maniera invisibile all'utente, e l'esecuzione salterà alla posizione indicata dall'etichetta **Gestione\_Errori**, ovvero sarà mostrato un avviso. Il codice relativo alla chiusura del file non sarà eseguito.

- `On Error Goto Gestione_Errori`  
`Open "C:\FILE_INESISTENTE" For Input As #1`  
`Close #1`  
`Gestione_Errori:`  
`If Err.Number = 53 Then`  
`MsgBox "Il file non esiste", vbCritical`  
`Else`  
`MsgBox "Errore: " & Err.Description, vbCritical`  
`End If`

In maniera analoga alla precedente al verificarsi dell'errore l'esecuzione salterà alla posizione indicata dall'etichetta **Gestione\_Errori**, in seguito alla quale il codice verificherà il codice dell'errore. In caso di errore 53 sarà mostrato un avviso, per tutti gli altri errori sarà mostrato un avviso differente con la descrizione dell'errore. Il codice relativo alla chiusura del file non sarà eseguito.

- On Error Resume Next  
Kill "C:\FILE\_INESISTENTE"  
On Error Goto 0  
Open "C:\FILE\_INESISTENTE" For Input As #1

L'operazione di cancellazione non sarà eseguita e nessun errore sarà mostrato all'utente; alla terza riga cesserà la gestione degli errori cosicché la successiva provocherà l'interruzione del programma e la chiusura con errore.

Ritornando al nostro progetto, abbiamo detto di voler ignorare il problema relativo all'uso del comando Incolla sulle caselle di testo rese numeriche; sappiamo già che se l'utente dovesse incollare un simbolo non numerico sarà generato errore. A questo scopo vogliamo proteggere l'intera routine relativa all'inserimento di un nuovo brano con un semplice gestore che rinvierà l'inserimento fintanto che persiste la condizione di errore:

```

1. Private Sub cmdAggiungiBrano_Click()
2.   If blnModifica = False Then
3.     ... Codice trattato nelle lezioni precedenti ...
4.   Else
5.     On Error Resume Next
6.     With udtDiscoCorrente
7.       .intIndice = LOF(intFileDati) / Len(udtDiscoCorrente) + 1
8.       .strTitoloBrano = txtTitoloBrano.Text
9.       .strAutore = txtAutoreBrano.Text
10.      .strTitoloCD = txtTitoloCD.Text
11.      .intTraccia = CInt(txtNumeroTraccia.Text)
12.      .intAnnoPubblicazione = CInt(txtAnnoPubblicazione.Text)
13.      .strNote = txtNote.Text
14.      .blnEliminato = False
15.      If Err.Number = 0 Then
16.        Put #intFileDati, .intIndice, udtDiscoCorrente
17.        lstBrani.AddItem Trim$(.strTitoloBrano)
18.        lstBrani.ItemData(lstBrani.NewIndex) = .intIndice
19.      Else
20.        MsgBox "Si è verificato un errore nell'inserimento dei dati", vbCritical
21.        lstBrani_Click
22.        Exit Sub
23.      End If
24.      On Error GoTo 0
25.    End With
26.    cmdAggiungiBrano.Caption = "&Aggiungi brano"
27.  End If
28.  blnModifica = Not blnModifica
29.  lstBrani.Enabled = Not blnModifica
30.  ... Codice trattato nelle lezioni precedenti ...
31. End Sub

```

Prima della fase di inserimento è stata aggiunto un gestore di errori che rinvierà procedendo alla riga successiva (riga 5); l'inserimento dei dati sarà effettuato soltanto se al termine del riempimento non risulta alcun errore (riga 15); viceversa sarà mostrato un avviso che informa della presenza di errori (riga 20) e sarà reinizializzata la variabile **udtDiscoCorrente** semplicemente chiamando la routine *lstBrani\_Click*, la routine legata all'evento Click sulla ListBox già trattata nelle lezioni precedenti. Al termine dell'operazione sarà richiesta l'uscita forzata dalla procedura, per non eseguire tutte quelle azioni legate al completamento.

Alla riga 24 il gestore di errori precedentemente avviato sarà annullato e tutti gli eventuali errori successivi non saranno gestiti. Questo approccio consente di non nascondere errori

non previsti ma di rivelarli immediatamente alla loro generazione.

La riga 29 è stata aggiunta per impedire lo spostamento della riga di puntamento al brano durante la fase di modifica.

In maniera analoga sarà modificata la routine relativa alla modifica di un brano esistente, non esposta in questa sede per brevità.

Esistono ancora una serie di possibili problemi da gestire ma che non saranno affrontati in questa sede; la maggior parte delle volte i programmi terminati richiedono ancora parecchie modifiche prima di divenire programmi completi. Spesso si tratta di gestire errori relativi a situazioni non previste o anomale e che durante l'analisi sfuggono per la complessità del problema o per semplice dimenticanza.

[Fibia FBI](#)

31 Dicembre 2003



[Torna alla nona lezione](#)

[Vai all'undicesima lezione](#)

