Corso di Visual Basic

(Parte 8)

DI MAURIZIO CRESPI

Questo mese il corso di programmazione in Visual Basic focalizza la propria attenzione sulle procedure, talvolta dette subroutine

L'oggetto dell'ottava puntata del corso dedicato alla programmazione in ambiente Microsoft Visual Basic è rappresentato dalle procedure definibili dal programmatore. Come si potrà osservare in seguito, si tratta di strutture di fondamentale importanza, in quanto il loro uso permette di rendere notevolmente più compatto e leggibile il codice di un'applicazione.

LE SOLUZIONI DEGLI ESERCIZI DELLA SCORSA LEZIONE

Come sempre, prima di affrontare i nuovi argomenti sarà fornito al lettore uno spunto per rinfrescare la propria memoria su quanto appreso nella scorsa lezione. L'occasione è data dalla correzione degli esercizi in essa proposti.

Primo esercizio

Il primo esercizio richiede la realizzazione di un programma in grado di simulare lo spoglio delle schede elettorali. Per ogni scheda, l'applicazione deve leggere il numero del candidato votato e redigere, a scrutinio terminato, una classifica in ordine decrescente. Il compito può essere svolto dalla seguente procedura, che è eseguita nella fase di caricamento di un form contenente un'etichetta di testo (*lblClassifica*) destinata a contenere la classifica finale.

```
Private Sub Form_load()
  Dim ElencoVoti() As Integer
  Dim Classifica() As Integer
  Dim Stringa As String
  Dim Candidato As Integer
  Dim NumeroCandidati As Integer
  Dim i As Integer
  Dim j As Integer
  Dim x As Integer
  Dim y As Integer

  Stringa = InputBox("Numero candidati")
  NumeroCandidati = Int(Val(Stringa))
  ReDim ElencoVoti(NumeroCandidati)
  ReDim Classifica(NumeroCandidati)

  Rem Acquisizione dei dati
  Do
    Stringa = InputBox("N. candidato votato
                                    (0 per finire)")
    Candidato = Int(Val(Stringa))
    If Candidato > 0 Then
      ElencoVoti(Candidato) = ElencoVoti(Candidato) + 1
    End If
  Loop Until Candidato = 0

  Rem Preparazione vettore classifica
  For i = 1 To NumeroCandidati
    Classifica(i) = i
  Next i
  For i = 1 To NumeroCandidati - 1
  For j = i + 1 To NumeroCandidati
    x = Classifica(i)
    y = Classifica(j)
    If ElencoVoti(x) < ElencoVoti(y) Then
      Temp = Classifica(i)
      Classifica(i) = Classifica(j)
      Classifica(j) = Temp
    End If
  Next j, i

  Rem Visualizzazione classifica
  lblClassifica.Caption = ""
  For i = 1 To NumeroCandidati
    lblClassifica.Caption = lblClassifica.Caption &
                            Classifica(i) & Chr$(10)
  Next i
End Sub
```

La lettura dei dati è, per semplicità, affidata a delle finestre di input, realizzate per mezzo della funzione *InputBox*.

Sono previsti due vettori; il primo, denominato *ElencoVoti*, associa all'elemento di indice *n* il numero dei voti collezionati dal candidato numero *n*. Ogni elemento contiene un numero che è incrementato ogniqualvolta è estratta una scheda sui cui è espressa una preferenza riferita al candidato corrispondente. Un secondo vettore, denominato *Classifica*, ha lo scopo di memorizzare i numeri associati ai candidati in modo che leggendo sequenzialmente la struttura si possa leggere la classifica in ordine decrescente di voti. Entrambi i vettori sono dimensionati all'avvio del programma per prevedere un elemento per ogni candidato.

La procedura provvede a leggere tutti i voti di preferenza, aggiornando volta per volta il vettore ad essi dedicato. Dopo aver fatto ciò, è inizializzato il vettore *Classifica* con i numeri corrispondenti a tutti i candidati; tali valori sono in seguito scambiati fra loro in modo tale per cui al termine del ciclo il vettore contenga la classifica in ordine decrescente di preferenze. Si noti che per effettuare ciò è applicata una variante all'algoritmo di ordinamento per selezione diretta visto nella scorsa lezione, in cui l'ordinamento è eseguito sull'array *Classifica* ma i confronti sono effettuati fra gli elementi del vettore *ElencoVoti*. Si desidera infatti fare in modo

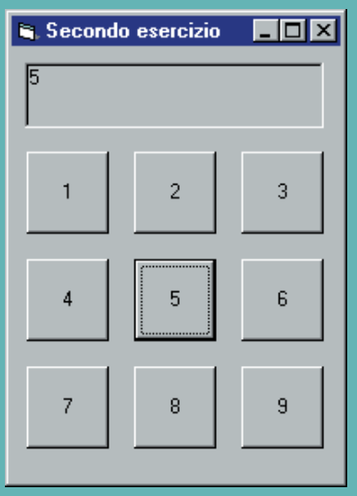

FIGURA 1

L'inter faccia utente della soluzione del secondo esercizio proposto nella scorsa lezione

che, dati due elementi qualsiasi *x* e *y* del vettore *Classifica*, *x* preceda *y* se *ElencoVoti(x)>ElencoVoti(y)*.

Secondo esercizio

Il secondo esercizio prevede la realizzazione di un programma dotato di una pulsantiera costituita da tasti numerati da 1 a 9. La pressione di uno di essi deve provocare l'incremento di un contatore numerico di un'entità pari all'indicazione presente sulla sua etichetta. Il valore del contatore deve essere visualizzato per mezzo di una label. Se si fa uso di un vettore di controlli, la soluzione diventa estremamente semplice, al punto da non necessitare di alcun commento. Si supponga di disegnare un form dotato di nove pulsanti, che costituiscono gli elementi del vettore *btnNumeri*. Alla pressione di uno di essi, ovvero al verificarsi dell'evento *Click* su un elemento del vettore, può essere associata la seguente procedura:

```
Private Sub btnNumeri_click(index As Integer)
    Dim Contatore As Integer
    Dim Incremento As Integer
    Contatore = Val(lblContatore.Caption)
    Incremento = Val(btnNumeri(index).Caption)
    Contatore = Contatore + Incremento
    lblContatore.Caption = Contatore
End Sub
```

LE PROCEDURE

Si supponga di voler realizzare un'applicazione in grado di calcolare la somma di due valori numerici inseriti dall'utente dopo aver verificato che siano positivi e, rispettivamente, inferiori a 10 e 50. Una possibile implementazione è la seguente:

```
Private Sub Form_Load()
  Dim Stringa As String
  Dim n1 As Double
  Dim n2 As Double
  Dim Somma As Double
  Rem Primo valore
  Do
     Stringa = InputBox("Primo valore:")
    n1 = Val(Stringa)
  Loop Until (n1 > 0) And (n1 < 10)
  Rem Secondo valore
  Do
     Stringa = InputBox("Secondo valore:")
    n2 = Val(Stringa)
  Loop Until (n2 > 0) And (n2 < 50)
  Somma = n1 + n2
  MsgBox ("La somma è " & Somma)
End Sub
```

L'acquisizione dei valori avviene per mezzo di due cicli. Il loro scopo consiste nel richiedere la digitazione di una stringa all'utente e nell'effettuare la conversione in un valore numerico per mezzo della funzione *Val*. Tale dato è successivamente sottoposto a verifica per determinarne la conformità alle condizioni richieste. Qualora essa non fosse riscontrata, la presenza del ciclo fa sì che sia richiesto nuovamente il dato all'utente. Si noti che le due strutture di iterazione sono estremamente simili. Ciò che cambia è rappresentato solamente dalla variabile da acquisire e dal valore massimo consentito. Se Visual Basic disponesse di un'istruzione in grado di richiedere un dato e di verificarne l'appartenenza ad un intervallo, il codice sarebbe notevolmente più semplice e privo di ripetizioni. Una simile istruzione non è prevista nella libreria standard dello strumento di sviluppo. Tuttavia, come tutti i linguaggi di programmazione moderni, Visual Basic permette la creazione di procedure, o *subroutine*. Esse rappresentano gruppi di istruzioni racchiuse in strutture identificate univocamente dai propri nomi. Una sequenza di questo tipo può essere eseguita più volte all'interno di un programma senza che sia necessario ripeterne il codice; basta infatti richiamarla indicandone il nome.

Mediante l'uso delle procedure è possibile migliorare la leggibilità del codice

La definizione di una procedura prevede la seguente sintassi:

```
[Public|Private] Sub <nome> [(<definizione_parametro_1>,
                             ... <definizione_parametro_n>)]
   [<dichiarazione_variabili_locali>]
   <istruzione_1>
   ...
   <istruzione_n>
End Sub
```

Dopo la parola chiave *Sub* è necessario specificare un identificatore che costituisce il nome della struttura. Come tutti gli altri identificatori, deve essere composto da lettere, numeri o dal carattere di sottolineatura (undescore) e il primo carattere deve essere necessariamente costituito da una lettera dell'alfabeto. Si noti che non è la prima volta che si usa la parola chiave *Sub* all'interno di questo corso. Infatti, le porzioni di codice da associare come risposta agli eventi sono anch'esse delle procedure. Quando un controllo genera un evento, l'interprete provvede automaticamente a verificare l'esistenza di una procedura avente un nome composto dall'identificatore del controllo seguito dal carattere di sottolineatura e dal tipo di evento (ad esempio *btnPulsante_Click*). Se tale struttura esiste, è eseguita. Le procedure oggetto di trattazione in questa lezione sono invece definite dall'utente con lo scopo di migliorare la leggibilità del programma e riciclare il codice già scritto, evitando la ripetizione di linee pressoché identiche. Le procedure, analogamente alle variabili, sono soggette alle regole di *scope*. Pertanto, se sono definite all'interno di un form, risultano locali a quest'ultimo, quindi

pressoché invisibili agli altri moduli. Qualora si desiderasse creare delle procedure globali, da utilizzare indifferentemente in tutti i form come se si trattasse di comandi standard previsti dal linguaggio di programmazione, è necessario creare un nuovo modulo globale, agendo sul menu *Progetto* alla voce *Inserisci modulo*. L'uso dei moduli globali permette anche di creare delle librerie di procedure che possono essere agevolmente riutilizzate per lo sviluppo di altre applicazioni, semplicemente includendo il file all'interno dei progetti.

UNA FUNZIONE IN UN MODULO

Per inserire una procedura in un modulo occorre selezionare il file nella finestra *Progetto* e premere il pulsante di visualizzazione del codice. A questo punto è possibile utilizzare la voce *Inserisci routine* del menu *Strumenti*. Appare così una finestra che permette di specificare il nome della procedura, unitamente ad altre opzioni che per ora non saranno descritte.

Agendo in questo modo sul form di avvio dell'applicazione descritta nell'esempio precedente, è possibile inserire in esso la procedura *LeggiValore*, in grado di leggere un dato e di convertirlo in un valore numerico.

La sua definizione è la seguente:

```
Sub  LeggiValore()
  Dim Stringa As String
  Do
    Stringa  =  InputBox(MessaggioRichiesta)
    n = Val(Stringa)
  Loop Until (n > 0) And (n < Massimo)
End Sub
```

Si noti la dichiarazione della variabile *Stringa*, che risulta locale alla procedura, essendo stata definita al suo interno. Pertanto, è creata quando il programma inizia ad eseguire la routine ed è distrutta in corrispondenza dell'uscita dal blocco di codice. Risulta quindi evidente che il suo valore non può essere utilizzato al di fuori della procedura. Inoltre, non è conservato nelle chiamate successive, a meno che non si sostituisca la parola chiave *Dim* con *Static*. In questo caso, la variabile è detta *statica*.

Tornando alla procedura *LeggiValore*, si nota che essa fa riferimento anche alle variabili *n*, *MessaggioRichiesta* e *Massimo*, di cui tuttavia non è specificata la definizione. Ciò è corretto se si tratta di due variabili globali, ovvero se sono definite all'interno della sezione *Generale* del form o in un modulo globale facendo uso della parola chiave *Global*. Nel primo caso esse risultano visibili a tutte le procedure definite all'interno del form.

Nel secondo, possono essere utilizzate addirittura da tutte le routine contenute nell'applicazione. Con l'introduzione della procedura *LeggiValore*, il codice da associare all'evento *Load* del form principale, diventa il seguente:

```
Private Sub Form_Load()
    Dim Stringa As String
    Dim n1 As Double
    Dim n2 As Double
    Dim Somma As Double
    Rem Primo valore
    MessaggioRichiesta = "Primo valore:"
    Massimo = 10
    LeggiValore
    n1 = n
    Rem Secondo valore
    MessaggioRichiesta = "Secondo valore:"
    Massimo = 50
    LeggiValore
    n2 = n
    Somma = n1 + n2
    MsgBox ("La somma è " & Somma)
End Sub
```

Si noti l'uso delle variabili globali per comunicare alla procedura il valore massimo accettabile, nonché il messaggio da visualizzare. Si noti altresì che la routine provvede ad utilizzare un'altra variabile globale (*n*) per restituire il valore letto. Questo primo esempio di utilizzo di una procedura ha sortito un effetto pressoché deludente, in quanto non ha comportato una sensibile semplificazione del codice. Ciò a causa della necessità di utilizzare delle variabili globali per scambiare le informazioni fra i due blocchi di istruzioni e dalla conseguente presenza di numerose operazioni di assegnamento. Al fine di ottenere un consistente miglioramento della leggibilità del codice, nonché una sua semplificazione, è necessario modificare la procedura *LeggiValore* per fare in modo che sia in grado di accettare dei parametri, ovvero che permetta di inizializzare le variabili *MessaggioRichiesta* e *Massimo* nel momento della sua invocazione, senza che sia richiesto l'uso delle istruzioni di assegnamento.

I PARAMETRI

Il metodo più semplice ed efficace per comunicare delle informazioni a una procedura consiste nell'uso dei parametri, ognuno dei quali deve essere dichiarato secondo la sintassi:

```
[ByVal|ByRef] <nome> As <Tipo>
```

Un parametro, talvolta detto argomento, per il codice contenuto all'interno della routine equivale a una variabile. Da essa si differenzia per il fatto che la sua inizializzazione non avviene all'interno della procedura, bensì quando essa è richiamata. Si supponga ad esempio di definire la seguente subroutine:

```
Sub Prova(Param1 As String, Param2 As Integer)
    MsgBox(Param1 & "-" & Param2)
End Sub
```

Se si digita la riga

```
Prova "Dev", 1998
```

si richiama la procedura *Prova* e si inizializzano i parametri *Param1* e *Param2* rispettivamente con la stringa "Dev" e con il numero 1998. Il messaggio visualizzato è pertanto il seguente:

```
Dev-1998
```

Si noti che un parametro può essere di un qualsiasi tipo base previsto da Visual Basic. Come accade per le variabili, se il formato non è dichiarato, il dato è considerato *Variant*.

La procedura *LeggiValore* può pertanto essere riscritta in modo da accettare come parametri il messaggio da visualizzare al momento della richiesta del dato e il valore massimo consentito.

```
Sub  LeggiValore(MessaggioRichiesta  As  String,
                                        Massimo  As  Integer)
  Dim Stringa As String
  Do
    Stringa  =  InputBox(MessaggioRichiesta)
    n = Val(Stringa)
  Loop Until (n > 0) And (n < Massimo)
End Sub
```

Per effetto di questa modifica, il codice da associare al caricamento del form principale si semplifica notevolmente:

```
Sub  Form_Load()
    Dim n1 As Double
    Dim n2 As Double
    Dim Somma As Double
    LeggiValore "Primo valore:", 10
    n1 = n
    LeggiValore "Secondo valore:", 50
    n2 = n
    Somma = n1 + n2
    MsgBox ("La somma è " & Somma)
End Sub
```

Le modifiche apportate fanno sì che non sia più necessario dichiarare le variabili globali *MessaggioRichiesta* e *Massimo*.

Un'ulteriore semplificazione potrebbe essere introdotta rendendo la procedura in grado di aggiornare automaticamente le variabili *n1* e *n2*. Ciò è possibile introducendo un terzo parametro. La procedura *LeggiValore* diventa pertanto la seguente:

```
Sub  LeggiValore(MessaggioRichiesta  As  String,
                              Massimo As Integer, n As Double)
  Dim Stringa As String
  Do
    Stringa  =  InputBox(MessaggioRichiesta)
    n = Val(Stringa)
  Loop Until (n > 0) And (n < Massimo)
End Sub
```

Il codice da eseguire in fase di caricamento del form diventa:

```
Sub  Form_Load()
  Dim n1 As Double
  Dim n2 As Double
  Dim Somma As Double
  LeggiValore "Primo valore:", 10, n1
  LeggiValore "Secondo valore:", 50, n2
  Somma = n1 + n2
  MsgBox ("La somma è " & Somma)
End Sub
```

L'applicazione ora non necessità più di alcuna variabile globale.

LE MODALITÀ DI PASSAGGIO DEI PARAMETRI

I parametri possono essere passati a una procedura secondo due diverse modalità: *per valore* e *per riferimento*; la scelta è effettuata utilizzando per ogni singolo parametro in alternativa le clausole *ByVal* o *ByRef*.

Passaggio dei parametri per riferimento

Nell'esempio precedente, la variabile *n* è stata passata alla procedura *LeggiValore* non per fornire ad essa un'informazione, bensì per fare in modo che sia la routine a restituire un valore al blocco di istruzioni chiamante. Infatti, nel momento in cui è eseguita la riga

```
LeggiValore "Primo valore:", 10, n1
```

alla variabile *n1* non è stato ancora assegnato un valore. Questo compito spetta alla procedura. Si dice allora che la variabile *n1* è stata passata *per riferimento*, ovverosia che è stato fornito alla procedura l'indirizzo della locazione di memoria in cui risiede *n1* per fare in modo che il dato in essa contenuto sia modificato. Questo è metodo di passaggio delle informazioni predefinito in Visual Basic. Quando alla dichiarazione di un parametro non è anteposta la clausola *ByVal*, esso è considerato dall'interprete come passato per riferimento. È quindi evidente che la clausola *ByRef* risulta di fatto superflua, in quanto ha solo lo scopo di rendere il codice più rapidamente comprensibile.

Passaggio dei parametri per valore

Se nella dichiarazione di un parametro si antepone la clausola *ByVal*, si stabilisce che l'informazione è passata *per valore*. Ciò significa che la procedura riceve un valore ma non la facoltà di variarlo. In pratica il dato deve essere considerato dalla routine alla stregua di una costante.

In realtà, la procedura può effettuare delle modifiche al valore del parametro, ma esse non hanno effetto nel blocco chiamante. Ad esempio, si supponga di dichiarare le seguenti procedure:

```
Sub  Prova1(ByVal  Testo  As  String)
  Testo = Testo & " 5.0"
  MsgBox Testo
End  Sub

Sub  Prova2(ByRef  Testo  as  String)
  Testo = Testo & " 5.0"
  MsgBox Testo
End  Sub
```

Si supponga altresì di dichiarare una variabile di tipo stringa denominata *Messaggio* e di inserirvi la frase "Visual Basic". La riga

```
Prova1  Messaggio
```

provoca pertanto la visualizzazione di una finestra in cui appare la scritta "Visual Basic 5.0". Il contenuto della variabile *Messaggio*, tuttavia, non varia, in quanto essa è stata passata per valore.

Al contrario, l'esecuzione della riga

```
Prova2  Messaggio
```

provoca la visualizzazione di una finestra perfettamente identica a quella descritta in precedenza, ma comporta anche la modifica del contenuto della variabile *Messaggio*.

DUE SEMPLICI ESERCIZI

Al fine di verificare la comprensione degli argomenti esposti, si provi a realizzare una procedura in grado di restituire il valore assoluto di un numero intero, usando a tal fine un parametro passato per riferimento. Si realizzi poi una seconda procedura che richiami la prima per calcolare la somma dei valori assoluti di 3 numeri passati come parametri per valore.

CONCLUSIONI

Un uso corretto delle procedure è di fondamentale importanza per la scrittura di codice leggibile e di facile manutenzione. Inoltre, il loro raggruppamento in librerie permette di creare dei moduli che possono essere riutilizzati nella realizzazione di applicazioni da parte di uno o più utenti come se si trattasse di estensioni del linguaggio di programmazione.

Maurizio Crespi si occupa principalmente di grafica e multimedia. Svolge la funzione di responsabile tecnico presso Datanord Multimedia, società specializzata nella realizzazione di software orientato al marketing e all'editoria, per la quale progetta e sviluppa applicazioni in C++, Visual Basic, Delphi e Director. Può essere contattato per e-mail come crespi@programmers.net.