Corso di Visual Basic (Parte V) *di Maurizio Crespi*

Oggetto di studio della quinta parte del corso dedicato alla programmazione in Visual Basic sono ancora i cicli. Questa volta l'obbiettivo è puntato sull'uso delle strutture regolate da una condizione booleana

La necessità di poter eseguire ripetitivamente un gruppo di istruzioni è sentita durante la stesura di pressoché qualsiasi applicazione non banale. Per questo motivo, tutti gli strumenti di sviluppo prevedono il supporto per le strutture di iterazione. Dopo aver soffermato l'attenzione sulla struttura *For*, si procederà ora a studiare l'uso dei cicli basati sulla parola chiave *Do*, che permettono di legare il numero delle ripetizioni non più al valore assunto da un contatore, bensì al verificarsi di una condizione logica.

Le soluzioni degli esercizi dello scorso numero

Come ormai consuetudine, la prima parte della lezione è dedicata alla correzione degli esercizi proposti nello scorso numero.

Primo esercizio

Il primo esercizio prevede la realizzazione di un programma che, dato in ingresso un numero intero, visualizzi la somma di tutti i numeri naturali pari minori o uguali ad esso. Dopo aver disegnato un form dotato di una casella di testo, a cui può essere assegnato il nome *txtNumero*, di una label (*lblRisultato*) e di un pulsante (*btnCalcola*), è possibile scrivere la procedura da associare alla pressione di quest'ultimo:

```
Private Sub btnCalcola_Click()

Dim Numero As Integer
Dim Contatore As Integer
Dim Somma As Long

Numero = Val(txtNumero.text)
If Numero > 0 Then
Somma = 0
    For Contatore = 2 To Numero Step 2
    Somma = Somma + Contatore
    Next Contatore

lblRisultato.Caption = Somma
Else
lblRisultato.Caption = "Numero negativo o nullo"
End If

End Sub
```

Il suo scopo è di trasformare in un dato numerico la stringa posta nella casella di testo e, se tale valore risulta positivo, di creare un ciclo per calcolare tutti i numeri interi pari minori ad esso. A tal fine fa uso di una struttura *For* avente come valore iniziale 2, ovvero il più piccolo numero pari, e come passo ancora 2. In tal modo, sono esclusi dal conteggio i valori dispari, cioè non multipli di 2. Il limite massimo è rappresentato dal numero fornito dall'utente. Si noti che, nel caso in cui quest'ultimo sia dispari, la struttura fa sì che il contatore assuma in realtà un valore massimo inferiore di un'unità rispetto a quello indicato. Tutti i valori assunti dal contatore sono aggiunti alla variabile *Somma*, che al termine delle iterazioni contiene il risultato desiderato.

Secondo esercizio

Il secondo esercizio richiede la creazione di un programma che, dato in ingresso un numero intero positivo, restituisca il più grande numero naturale per cui esso è divisibile. Supponendo di riutilizzare il form descritto in precedenza, è possibile associare al pulsante la seguente procedura:

```
Private Sub btnCalcola_click()

Dim Contatore As Integer
Dim Divisore As Integer
Dim Numero As Double

Numero = Val(txtNumero.Text)
If Numero > 0 Then

     If Numero = Int(Numero) Then
     Divisore = 1
         For Contatore = 2 To (Numero - 1)
             If (Numero Mod Contatore) = 0 Then
               Divisore = Contatore
             End If
         Next Contatore
     lblRisultato.Caption = Divisore
     Else
     lblRisultato.Caption = "Numero non intero"
     End If

Else
lblRisultato.Caption = "Numero negativo o nullo"
End If

End Sub
```

Si fa ancora uso di un ciclo *For* in cui il contatore è inizializzato a 2 (è superfluo verificare la divisibilità per 1). Il valore finale è pari al numero diminuito di un'unità. Al il programma accede solo se il dato fornito dall'utente, memorizzato nella variabile *Numero*, risulta intero positivo. Ad ogni iterazione, si fa uso dell'operatore *Mod* per verificare se il numero fornito in ingresso è

divisibile per il valore della variabile *Contatore*. In caso affermativo, quest'ultimo è memorizzato nella variabile *Divisore* che, completato il ciclo, contiene il risultato desiderato.

Il comando Exit For

Si noti che, con la procedura sopra descritta, il numero delle ripetizioni aumenta all'aumentare del valore fornito in ingresso dall'utente. L'applicazione di tale algoritmo a un numero molto grande potrebbe pertanto causare, almeno in linea di principio, dei problemi di prestazioni. E' possibile scrivere una procedura più efficiente realizzando un ciclo con decremento. In questo caso, la struttura *For* ha come valore iniziale il dato fornito in ingresso diminuito di un'unità.

```
Private Sub btnCalcola_click()

Dim Contatore As Integer
Dim Divisore As Integer
Dim Numero As Double

Numero = Val(txtNumero.Text)
If Numero > 0 Then

  If Numero = Int(Numero) Then
    Divisore = Numero
         For Contatore = (Numero - 1) To 1 Step -1
                 If (Numero Mod Contatore) = 0 Then
                    Divisore = Contatore
         Exit For
                 End If
  Next Contatore
  lblRisultato.Caption = Divisore
  Else
  lblRisultato.Caption = "Numero non intero"
  End If
Else
lblRisultato.Caption = "Numero negativo o nullo"
End If

End Sub
```

Ad ogni iterazione è effettuato un controllo atto a stabilire se il contenuto della variabile *Contatore* è in grado di dividere il numero fornito dall'utente. Il primo valore di cui è rilevata la capacità di soddisfare questa condizione è sicuramente il maggior divisore. Una volta trovatolo, è pertanto inutile proseguire con le iterazioni. Per provocare l'uscita prematura dal ciclo, si fa uso del comando *Exit For*. In questo caso, il numero delle ripetizioni è quello strettamente necessario. Il vantaggio in termini di efficienza può essere notevole. E' tuttavia importante non abusare dell'uso di questo comando, in quanto ha influenza negativa sulla leggibilità del codice.

Il ciclo While

Una soluzione più elegante prevede l'uso del ciclo *While*, che permette la ripetizione di un segmento di codice per tutto il tempo in cui una condizione risulta vera. La sua sintassi è la seguente:

```
While <condizione>
   <istruzione 1>
   <istruzione 2>
   ...
   <istruzione n>
Wend
```

Le istruzioni comprese fra le parole chiave *While* e *Wend* sono ripetute per un numero di volte non stabilito rigidamente a priori, bensì dipendente dalle stesse istruzioni, che devono essere in grado di fare in modo che la condizione ad un certo punto smetta di verificarsi. In caso contrario, si incappa in un errore molto diffuso fra i programmatori alle prime armi, ovvero si crea ciò che è usualmente detto *ciclo infinito*. Gli effetti di un'iterazione senza fine sono evidenti. Il programma di fatto si blocca e per terminarlo occorre far ricorso al *task manager* del sistema operativo.

La procedura che costituisce la soluzione del secondo esercizio può essere riscritta utilizzando la struttura *While* nel modo seguente:

```
Private Sub btnCalcola_click()

Dim Contatore As Integer
Dim Divisore As Integer
Dim Numero As Double

Numero = Val(txtNumero.Text)

If Numero > 0 Then

     If Numero = Int(Numero) Then
     Contatore = Numero – 1
           While (Numero Mod Contatore) <> 0
           Contatore = Contatore – 1
           Wend
     lblRisultato.Caption = Contatore
     Else
     lblRisultato.Caption = "Numero non intero"
     End If

Else
lblRisultato.Caption = "Numero negativo o nullo"
End If

End Sub
```

Il suo funzionamento è estremamente semplice: un contatore è inizializzato al massimo numero intero inferiore al dato fornito in ingresso ed è decrementato all'interno del ciclo più volte per tutto il tempo che il resto della divisione per il contatore del numero digitato dall'utente si mantiene diverso da zero. Quando il resto si annulla, ovvero quando la variabile *Contatore* contiene il divisore desiderato, la condizione

```
(Numero Mod Contatore)<>0
```

diventa falsa e le iterazioni terminano. Il risultato può quindi essere visualizzato per mezzo dell'apposita label. Si noti che il ciclo non può mai essere infinito. La condizione infatti assume sicuramente un valore falso quando la variabile *Contatore* vale 1.

Le parole chiave Do e Loop

Il ciclo *While* può anche essere descritto in modo più elegante per mezzo delle parole chiave *Do* e *Loop*. In questo caso la sintassi diventa:

```
Do While <condizione>
  <istruzione 1>
  <istruzione 2>
  ...
  <istruzione n>
Loop
```

Il comportamento è analogo a quello visto in precedenza: le istruzioni contenute all'interno della struttura sono ripetute fintanto che la condizione indicata accanto alla parola *While* si verifica. La procedura dell'esempio precedente può quindi essere riscritta come segue:

```
Private Sub btnCalcola_click()

Dim Contatore As Integer
Dim Divisore As Integer
Dim Numero As Double

Numero = Val(txtNumero.Text)

If Numero > 0 Then

     If Numero = Int(Numero) Then
     Contatore = Numero – 1
           Do While (Numero Mod Contatore) <> 0
           Contatore = Contatore – 1
           Loop
     lblRisultato.Caption = Contatore
     Else
     lblRisultato.Caption = "Numero non intero"
     End If
```

```
Else
lblRisultato.Caption = "Numero negativo o nullo"
End If

End Sub
```

Il ciclo Do Until

Pressoché analogo è il ciclo *Do Until*, caratterizzato dalla seguente sintassi:

```
Do Until <condizione>
  <istruzione 1>
  <istruzione 2>
  ...
  <istruzione n>
Loop
```

In questo caso, le iterazioni avvengono quando la condizione è falsa e terminano quando essa si avvera. Il ciclo su cui si basa la soluzione del secondo esercizio può essere riscritto utilizzando la struttura *Do Until* nel modo seguente:

```
Do Until (Numero Mod Contatore) = 0
   Contatore = Contatore – 1
Loop
```

Si noti che l'uso di questa struttura in alternativa alla precedente non presenta dei vantaggi significativi. La scelta può pertanto essere effettuata caso per caso in base alla comprensibilità del codice e alle proprie preferenze personali.

Si consideri ora il seguente esempio. Si desidera realizzare un programma che, ricevendo in ingresso due stringhe, sia in grado di indicare la posizione in cui la seconda è eventualmente contenuta nella prima. In pratica, si tratta di realizzare una procedura in grado di simulare il comportamento della funzione *Instr*.

Per prima cosa si provvede a disegnare un form dotato di due caselle di testo, a cui è possibile dare i nomi *txtStringa* e *txtDaCercare*, e di un pulsante, a cui è dato il nome *btnCerca*. Ad esso è possibile associare la procedura che effettua la ricerca di una stringa nell'altra. Il codice è il seguente:

```
Private Sub btnCerca_click()

Dim Stringa As String
Dim DaCercare As String
Dim LunghStringa As Integer
Dim LunghDaCercare As Integer
Dim Posizione As Integer
Dim Trovato As Boolean
```

```
Stringa = txtStringa.Text
DaCercare = txtDaCercare.Text
LunghStringa = Len(Stringa)
LunghDaCercare = Len(DaCercare)
Posizione = 0
Trovato = False

  Do Until Trovato Or (Posizione + LunghDaCercare) > LunghStringa
   Posizione = Posizione + 1
   Trovato = (Mid$(Stringa, Posizione, LunghDaCercare) = DaCercare)
  Loop

If Trovato Then
   lblMessaggio.Caption = "Posizione = " & Str$(Posizione)
Else
   lblMessaggio.Caption = "Stringa non trovata"
End If

End Sub
```

La procedura dapprima calcola la lunghezza della stringa da cercare, poi estrae tutte le possibili sequenze di tale dimensione dal testo posto nella casella *txtStringa*. Il processo termina quando è estratta una sequenza uguale a quella oggetto di ricerca; in questo caso la variabile *Trovato*, contenente il risultato del confronto, assume il valore logico *True*. La condizione

```
Trovato Or (Posizione + LunghDaCercare) > LunghStringa
```

pertanto si verifica e il ciclo si conclude. La variabile *Posizione*, usata come contatore, contiene allora la posizione in cui la sequenza è stata localizzata. L'informazione in essa contenuta rappresenta il dato che il programma deve visualizzare per mezzo della label.

Come si può facilmente notare, le iterazioni possono terminare anche quando il numero dei caratteri presenti nella stringa a partire dalla posizione indicata dal contatore risulta inferiore alla lunghezza della sequenza da cercare. In questo caso, il ciclo termina senza che sia stata trovata alcuna occorrenza; tale eventualità provoca la visualizzazione di un opportuno messaggio di avviso per mezzo della label. Si noti che se la stringa da cercare ha lunghezza inferiore a quella entro cui deve essere effettuata la ricerca, le istruzioni poste all'interno del ciclo non sono mai eseguite.

Esercizio

Si provi a realizzare un'applicazione che, ricevuta in ingresso una stringa alfabetica, indichi la posizione in essa occupata dalla prima lettera maiuscola, se presente.

Ripetizione di un blocco di istruzioni per almeno una volta

Spesso si rivela utile poter fare in modo che, indipendentemente dal verificarsi della condizione che regola il ciclo, il blocco di istruzioni in esso contenuto sia eseguito almeno una volta. Ad esempio, si supponga di voler realizzare un'applicazione in grado di estrarre a sorte dei numeri compresi fra 1 e 10 mentre la loro somma si mantiene inferiore a 15. A tal fine si utilizza la funzione *Rnd*, che restituisce un numero pseudocasuale (ovvero generato in modo da sembrare estratto a sorte) compreso fra 0 e 1. E' evidente che almeno una volta deve essere effettuata l'estrazione di un numero. Per fare in modo che ciò avvenga, è possibile posizionare la condizione che regola il ciclo alla fine di esso anziché all'inizio. In questo caso la sintassi diventa:

```
Do
  <istruzione 1>
  <istruzione 2>
  ...
  <istruzione n>
Loop Until <condizione>
```

Per consentire all'estrazione di avere inizio in seguito alla pressione del pulsante *btnEstrai* e per far sì che i numeri generati siano visualizzati in una textbox, denominata *txtNumeri*, è necessario scrivere la seguente procedura:

```
Private Sub btnEstrai_Click()

Dim Somma As Integer
Dim Numero As Integer

Randomize
Somma = 0
txtNumeri.Text = ""

Do
  Numero = Int(10 * Rnd) + 1
  Somma = Somma + Numero
  txtNumeri.Text = txtNumeri.Text + Str$(Numero) + "-"
Loop Until Somma > 15

End Sub
```

Il ciclo provvede a calcolare un numero casuale compreso fra 1 e 10 per mezzo della funzione *Rnd*, moltiplicando il valore da essa restituito per 10, estraendone la parte intera e sommando il valore 1. Siccome, come già accennato in precedenza, la funzione restituisce un numero decimale maggiore o uguale a 0 e minore di 1, la parte intera del prodotto risulta compresa fra 0 e 9. Per fare in modo che alla variabile *Numero* sia assegnato un valore compreso fra 1 e 10 occorre pertanto aggiungere un'unità. Il valore ottenuto è aggiunto alla variabile *Somma*. I numeri sono visualizzati, separati da un trattino, nella textbox *txtNumeri*. Le iterazioni terminano quando il valore della variabile *Somma* diventa maggiore di 15.

Si noti l'uso dell'istruzione *Randomize* all'inizio della procedura. Esso ha lo scopo di inizializzare il generatore di numeri pseudocasuali. In assenza di tale operazione, i valori generati sarebbero sempre gli stessi ogni volta che è lanciato il programma.
Anche quando la condizione è posta alla fine del ciclo è possibile sostituire la parola chiave *Until* con *While*. In questo caso, le iterazioni avvengono fintanto che si verifica la condizione indicata.

Esercizio

Per valutare la propria comprensione degli argomenti trattati, si realizzi un'applicazione che, ricevendo in ingresso una stringa per mezzo di una textbox, restituisca la lunghezza della prima parola in essa contenuta.

Conclusioni

I calcolatori sono nati allo scopo di agevolare l'utente nello svolgimento delle operazioni ripetitive. Per questo motivo, le strutture che permettono di eseguire delle iterazioni rivestono una notevole importanza, tale da rendere la loro conoscenza imprescindibile per qualsiasi programmatore. Ancora una volta, pertanto, è rivolto al lettore l'invito ad esercitarsi per acquisire la necessaria dimestichezza con i concetti esposti.

Esercizio sulla ricerca della prima lettera maiuscola

```
Private Sub btpCalcola_Click()

Dim Stringa As String
Dim LunghStringa As Integer
Dim Posizione As Long
Dim Trovato As Boolean

Stringa = txtStringa.Text
LunghStringa = Len(Stringa)
Posizione = 0
Trovato = False
  Do Until Trovato
   Posizione = Posizione + 1
   valore = Mid$(Stringa, Posizione, 1)
      If valore = " " Then
      Else
         Trovato = (Mid$(Stringa, Posizione, 1) = UCase(valore))
      End If
  Loop
If Trovato Then
   lblMessaggio.Caption = "Posizione = " & Str$(Posizione)
   lblMessaggio1.Caption = "lettera maiuscola = " & valore
Else
   lblMessaggio.Caption = "Posizione non trovata"
End If

End Sub
```

```
Private Sub btpLunghezza_Click()

Dim Stringa As String
Dim LunghStringa As Integer
Dim Posizione As Long
Dim Trovato As Boolean

Stringa = txtStringa1.Text
LunghStringa = Len(Stringa)
Posizione = 0
Trovato = False
  Do Until Trovato
   Posizione = Posizione + 1
   valore = Mid$(Stringa, Posizione, 1)
      If valore = " " Then
         Trovato = True
      Else
         Trovato = (Mid$(Stringa, 1, Posizione) = " ")
      End If
  Loop
If Trovato Then
   lblMessaggio2.Caption = "Lunghezza prima parola = " & Str$(Posizione - 1)
   lblMessaggio3.Caption = "prima parola = " & Mid$(Stringa, 1, Posizione - 1)
Else
   lblMessaggio.Caption = "Posizione non trovata"
End If
End Sub
```