

Un programma di esempio

Dopo tanta teoria e tanti piccoli frammenti di codice, non si poteva non presentare almeno un programma di esempio scritto interamente in Visual Basic 2005. Questo progetto permette di sperimentare molte caratteristiche del linguaggio, come l'uso degli array, e la programmazione visuale, con l'utilizzo di finestre comuni di dialogo per il salvataggio e il recupero di file di testo e l'anteprima di stampa. Infine, sarà sfruttata una speciale caratteristica, costituita da un metodo della classe `Regex` contenuta nel namespace `System.Text.RegularExpressions`. Questo namespace tratta le cosiddette espressioni regolari che non sono state trattate in questo libro ma alle quali sarà dedicato un breve paragrafo nel presente capitolo. In determinati contesti le espressioni regolari sono estremamente importanti e permettono di ottenere risultati molto sofisticati.

Ancora una volta, per evitare che si possano avere dubbi un merito, si vuole far osservare che il progetto che viene presentato in questo capitolo può essere creato con qualsiasi versione di Visual Basic 2005 o di Visual Studio 2005. Pertanto, chi volesse riprodurre i passi, potrà farlo con la propria versione preferita.

Introduzione al progetto

Il progetto che sarà presentato in questo capitolo riguarda la creazione di un programma che simula una calcolatrice in grado di eseguire somme, sottrazioni, divisioni e moltiplicazioni.

Per far sì che non sembri il solito banale progetto di calcolatrice, si creerà anche una funzionalità un po' più interessante: sarà infatti aggiunta una parte grafica che riproduce il nastro di carta su cui vengono stampati gli operandi, gli operatori e i risultati.

Inoltre sarà possibile salvare una "strisciata" in un file di testo a propria scelta, rileggere il file di testo e ricaricarlo nel nastro di carta e stampare un'intera strisciata sulla stampante predefinita.

I calcoli saranno eseguiti approssimativamente secondo la modalità utilizzata nelle calcolatrici da tavolo.

Somma e sottrazione

Supponendo l'inserimento di una somma tra due numeri, prima viene inserito il primo addendo e premuto il tasto `+`, poi viene inserito il secondo addendo, viene premuto nuovamente il tasto `+` e infine viene premuto il tasto `=`. Per la sottrazione si procede in modo analogo, ma riportando il valore da sottrarre come secondo termine:

```

125 +      125 +
 64 +      64 -
--- =      --- =
189 T      61 T
=====

```

Moltiplicazioni e divisioni

Per le moltiplicazioni la procedura è sostanzialmente simile e cioè: prima viene inserito il primo operando e premuto il tasto +, poi viene inserito il secondo operando e premuto il tasto * (asterisco) e infine viene premuto il tasto =. Per le divisioni la procedura è analoga, con l'utilizzo del simbolo / (barra):

```

440 +      440 +
  2 *      5 /
--- =      --- =
880 T      88 T
=====

```

Il calcolo viene effettuato sul valore attualmente presente: ecco perché, per esempio, non è possibile anticipare l'operatore di moltiplicazione o di divisione:

```

100 +
100 -
--- =
  0 t
=====
440 *      <-- in questo punto, il valore 440 viene moltiplicato
  2 +      per il valore zero già presente
--- =
  2 T      <-- risultato = 2, sommato al precedente valore nullo
=====

```

Creazione del progetto

Per prima cosa si dovrà procedere alla creazione di un nuovo progetto:

1. avviare Visual Basic 2005 o Visual Studio 2005;
2. selezionare il menu *File > Nuovo > Progetto...*;
3. fare clic sul progetto *Applicazione Windows* (in Visual Studio si trova nel gruppo di progetti *Visual Basic > Windows*);
4. modificare il nome del progetto in *Calcolatrice* e il percorso come desiderato;
5. fare clic sul pulsante *OK* per confermare la creazione del progetto.

Il form Calcolatrice.vb

Il progetto presenta già un form predefinito di nome `Form1`. In questo paragrafo si personalizzerà il form per definire l'interfaccia grafica dell'applicazione che, al termine della procedura, sarà simile a quella mostrata nella Figura 12.1.

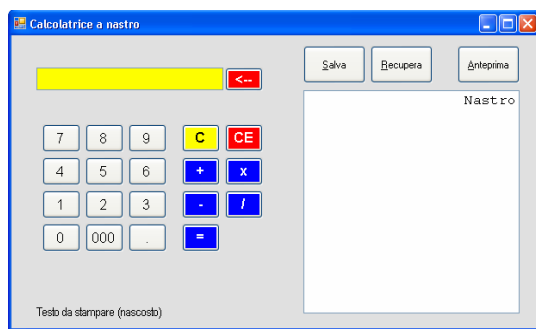


Figura 12.1 Il form dell'applicazione *Calcolatrice a nastro*.

Per la personalizzazione dell'interfaccia, saranno forniti i passi necessari a creare le varie parti del form e un elenco di proprietà che dovranno essere modificate come indicato.

Per prima cosa si modificano alcune proprietà del form stesso:

1. nel riquadro *Esplora Soluzioni*, fare clic con il pulsante destro del mouse sulla voce *Form1.vb* e selezionare la voce di menu *Rinomina*. Sostituire il nome *Form1.vb* con *Calcolatrice.vb*;
2. fare clic in qualsiasi punto del form e nella *Finestra delle Proprietà* selezionare la proprietà *Size* e inserire quanto segue: "620; 370";
3. cliccare poi sulla proprietà *Text* e inserire il testo *Calcolatrice a nastro*;
4. infine nella proprietà *StartPosition* indicare *CenterScreen*, per centrare il form sullo schermo.

Si inizi quindi ad inserire i vari controlli visuali e non visuali sul form, come indicato qui di seguito.

TextBox "Display"

In questo *TextBox* saranno inseriti i numeri da utilizzare nei calcoli. L'inserimento potrà essere effettuato digitando direttamente le cifre sulla tastiera oppure eseguendo dei clic con il mouse sui pulsanti del tastierino visualizzato a video.

Questo controllo visualizzerà anche i risultati dei calcoli, dopo aver premuto il pulsante con il segno di uguale = oppure premendo il tasto *Invio*.

Nella Tabella 12.1 vengono presentati i valori da inserire nelle rispettive proprietà, per completare la personalizzazione del controllo. Si procederà così anche con i controlli successivi.

Tabella 12.1 – Proprietà del *TextBox Display*.

Proprietà	Valore
Name	Display
TextAlign	Right
Font	Arial; 12pt
BackColor	Yellow
ForeColor	Black
Size	217; 26
Location	29; 37
ReadOnly	True

Pulsante "TastoCancellaUno"

A destra del *TextBox* è presente un pulsante che permette di cancellare la cifra appena inserita, cioè la cifra meno significativa.

Tabella 12.2 – Proprietà del pulsante *TastoCancellaUno*.

Proprietà	Valore
Name	TastoCancellaUno
Text	<--

Font	Arial; 12pt; style=Bold
BackColor	Red
ForeColor	White
Size	44; 27
Location	248; 37
UseVisualBackColor	False

Pulsanti 1..0, punto decimale e 000

Nell'area inferiore al *TextBox Display* è presente il tastierino numerico. Ciascun tasto numerico inserisce la corrispondente cifra nel display, il tasto con il punto inserisce un punto decimale, mentre il tasto *000* inserisce tre zeri. Nelle tabelle seguenti saranno indicate le proprietà dei soli pulsanti 7, 4 e 8, perché le proprietà degli altri pulsanti numerici possono essere desunte per analogia, mentre saranno fornite anche le proprietà dei pulsanti relativi al punto decimale e ai tre zeri.

Tabella 12.3 – Proprietà di alcuni pulsanti del tastierino numerico.

Proprietà	Valore
<i><u>Per tutti i pulsanti:</u></i>	
Font	Arial; 12pt
UseVisualBackColor	False
Size	44; 32
<i><u>Pulsante "7":</u></i>	
Name	Tasto7
Text	7
Location	36; 102
<i><u>Pulsante "8":</u></i>	
Name	Tasto8
Text	8
Location	86; 102
<i><u>Pulsante "4":</u></i>	
Name	Tasto4
Text	4
Location	36; 140
<i><u>Pulsante ".":</u></i>	
Name	PuntoDecimale
Text	.
Location	136; 216
<i><u>Pulsante "000":</u></i>	
Name	Tasto00
Text	000
Location	86; 216

Pulsanti di cancellazione parziale e totale e tasti di operazione

Sul lato destro del tastierino numerico, come nella maggior parte delle calcolatrici, sono presenti alcuni pulsanti per la cancellazione dell'ultimo valore inserito (C), per la cancellazione totale (CE) e tutti quelli necessari all'esecuzione delle operazioni (+, -, *, /, =).

Il tasto C cancella il valore presente sul display, senza influire sul calcolo in corso, né sulle operazioni riportate sul nastro di carta.

Il tasto CE cancella il valore presente sul display, il calcolo in corso e anche il nastro di carta. La calcolatrice si presenterà come appena avviata.

Il tasto uguale fornisce il risultato sul display e anche sul nastro di carta, con opportune linee di separazione per evidenziare il totale stampato.

Anche in questo caso si forniscono le proprietà specifiche di ciascun pulsante e le proprietà di posizionamento e dimensione solo dei primi due pulsanti (C e CE) in quanto gli altri pulsanti sono inseriti con una uguale dimensione e ad una distanza regolare.

Tabella 12.4 – Proprietà dei pulsanti di cancellazione e di operazione.

Proprietà	Valore
<i><u>Per tutti i pulsanti:</u></i>	
Size	44; 32
UseVisualBackColor	False
<i><u>Pulsante "C":</u></i>	
Name	TastoC
Text	C
Font	Arial; 12pt; style=Bold
Location	198; 102
BackColor	Yellow
ForeColor	Black
<i><u>Pulsante "CE":</u></i>	
Name	TastoCE
Text	CE
Font	Arial; 12pt; style=Bold
Location	248; 102
BackColor	Red
ForeColor	White
<i><u>Pulsanti +, -, *, /, =:</u></i>	
Name	TastoPiu, TastoPer, TastoMeno, TastoDiviso, TastoUguale
Text	+, *, -, /, =
Font	Arial; 12pt; style=Bold
BackColor	Blue
ForeColor	White

Pulsanti per la gestione del nastro di carta

Nella parte destra del form sono presenti tre pulsanti per la gestione delle informazioni riportate dal nastro di carta:

- un pulsante per salvare il contenuto del nastro su file di testo. Una volta salvato il file su disco, il contenuto del nastro di carta viene cancellato;
- un pulsante per leggere e caricare un file di testo nel nastro di carta. L'eventuale contenuto del nastro di carta al momento del caricamento dal file di testo viene perso;
- un pulsante per aprire un form di anteprima di stampa per visualizzare la stampa del nastro di carta prima di effettuare la stampa effettiva sulla stampante predefinita sul sistema. Il contenuto del nastro di carta rimane invariato.

Tabella 12.5 – Proprietà dei pulsanti di gestione del nastro di carta.

Proprietà	Valore
Name	TastoSalva, TastoRecupera, TastoAnteprima
Text	&Salva, &Recupera, &Anteprima
Location	339; 12 417; 12 518; 12
Font	Arial; 9pt
Size	72; 43
UseVisualBackColor	False
BackColor	224; 224; 224
ForeColor	Black

Nota

Si richiama l'attenzione sulla proprietà `Text` impostata per i pulsanti di gestione del nastro di carta: il simbolo "&" inserito nel testo da visualizzare su ciascun pulsante indica che il carattere successivo dovrà essere sottolineato. Questa non è una semplice caratteristica estetica, ma secondo gli standard di Windows permette di premere un pulsante anche con la combinazione di tasti *ALT* + [tasto sottolineato]. Ad esempio per il pulsante Salva (proprietà `Text` = `&Salva`), si potrà premere tale pulsante anche con la combinazione *ALT+S*.

ListBox Nastro di carta

Nella parte inferiore destra del form è presente il nastro di carta, o meglio una *ListBox* che rappresenta un nastro di carta. Le proprietà di tale controllo sono quelle indicate nella Tabella 12.6.

Tabella 12.6 – Proprietà del nastro di carta.

Proprietà	Valore
Name	Nastro
Font	Courier New; 12pt
BackColor	White

ForeColor	Black
RightToLeft	Yes
Location	339; 63
Size	251; 256

Label accessoria

Nella parte inferiore del form, infine, è stata inserita una *Label* non visibile all'utente che serve come appoggio per la composizione del testo, come si vedrà poi dal codice sorgente.

Per questo controllo le proprietà *Size*, *Position*, *Text*, *BackColor*, *ForeColor*, *Font* e così via non hanno importanza, dato che tale controllo non è visualizzato. Si esamineranno, pertanto, solamente le proprietà che sono elencate nella Tabella 12.7.

Tabella 12.7 – Proprietà della *Label* accessoria nascosta.

Proprietà	Valore
Name	TestoDaStampare
Visibile	False

Finestre comuni di dialogo

Ovviamente in questo progetto sono state inserite anche delle finestre comuni di dialogo per fornire le funzionalità di salvataggio su file, caricamento da file e anteprima di stampa. Le proprietà assegnate a tali controlli, visibili solo nella parte inferiore della finestra di progettazione delle form, sono quelle elencate nella Tabella 12.8.

Tabella 12.8 – Proprietà delle finestre comuni di dialogo.

Proprietà	Valore
<u>Controllo SaveFileDialog:</u>	
Name	SaveFileDialog1
Filter	File di testo (*.txt) *.txt
<u>Controllo OpenFileDialog:</u>	
Name	OpenFileDialog1
Filter	File di testo (*.txt) *.txt
<u>Controllo PrintPreviewDialog:</u>	
Name	PrintPreviewDialog1

Espressioni regolari

Prima di introdurre la spiegazione del codice sorgente, si vuole aprire una parentesi sulle espressioni regolari, dato che queste sono utilizzate nel codice. L'utilizzo che ne viene fatto nel programma è molto limitato, ma la funzionalità utilizzata è interessante e riutilizzabile in altri contesti della programmazione.

Cenni sulle espressioni regolari

Le espressioni regolari sono progettate per effettuare analisi e modifiche del testo. Per utilizzare le espressioni regolari esistono delle regole ben precise e una sintassi che nell'insieme costituiscono il linguaggio delle espressioni regolari. Il linguaggio utilizza due tipi fondamentali di caratteri: caratteri normali e metacaratteri. La capacità di elaborazione delle espressioni regolari deriva dall'insieme di metacaratteri.

Per fare un esempio di metacaratteri, si pensi ai "caratteri jolly" utilizzati con il sistema operativo MS-DOS: il carattere ? viene utilizzato come segnaposto per un carattere qualsiasi, mentre * viene utilizzato come segnaposto per un numero imprecisato di caratteri (nessuno, uno o molti).

In DOS, il comando `COPY *.DOC A:` consente di copiare tutti i file con estensione DOC sul disco nell'unità A. Il metacarattere * indica qualsiasi nome di file davanti all'estensione DOC.

Le espressioni regolari espandono molto questo concetto, poiché forniscono un insieme estremamente ampio di metacaratteri in grado di descrivere espressioni per criteri di ricerca di testo complesse con pochi caratteri.

Ad esempio l'espressione regolare `\s2006`, quando viene applicata ad un testo, viene soddisfatta da tutte le occorrenze della stringa "2006" preceduta da un carattere di spazio o di tabulazione.

Con le espressioni regolari si possono eseguire ricerche anche più complesse. Ad esempio, l'espressione regolare `(?<char>\w)\k<char>` consente di cercare i caratteri abbinati adiacenti. Se applicata alla stringa "Tutti sono invitati a leggere questo libro!", trova le corrispondenze nelle parole "Tutti" e "leggere".

E' possibile utilizzare le espressioni regolari in Visual Basic 2005 in due modi diversi:

- inserendo un'istruzione `Imports System.Text.RegularExpressions` all'inizio di ogni modulo o classe che ne utilizza un'istanza;
- impostando il riferimento al namespace `System.Text.RegularExpressions` direttamente nella configurazione del progetto: nella finestra *Esplora Soluzioni*, fare doppio clic su *My Project*, selezionare la scheda *Riferimenti* e spuntare la casella di controllo corrispondente al namespace sopra indicato. A questo punto il namespace sarà disponibile per tutti i moduli presenti attualmente o che saranno creati nel progetto.

La classe clsNastro.vb

Il form *Calcolatrice.vb* crea un'istanza della classe *clsNastro* perché quasi tutte le azioni che possono essere svolte attraverso i controlli del form modificano lo stato di tale oggetto.

Pertanto, prima di analizzare il codice del form, è opportuno esaminare il contenuto della classe *clsNastro* per poter comprendere come è stato definito il suo comportamento.

Per prima cosa si veda l'intestazione della classe e la sezione dichiarativa:

```
Public Class clsNastro
    Private arrayNastro(0) As String
    Private ctrl As New ListBox
```

Il primo elemento (elemento zero) dell'array non viene mai utilizzato: in questo modo è possibile capire che il nastro di carta deve essere svuotato quando l'array viene ridimensionato a un elemento (cioè il solo elemento vuoto, l'elemento zero).

Dopo l'intestazione della classe, vengono dichiarate due variabili private e quindi non accessibili direttamente dall'esterno della classe:

- un array di stringhe, `arrayNastro()`, che conterrà le singole righe del nastro di carta;
- una variabile oggetto di tipo *ListBox*, `ctrl`, che sarà utilizzata per conservare un riferimento alla *ListBox* presente nella form. Tale variabile permette alla classe di leggere, inserire e

modificare il contenuto della *ListBox* ed è valorizzata già nel costruttore della classe, come è possibile vedere dal seguente codice:

```
Public Sub New(ByVal controllo As ListBox)
    ctrl = controllo
End Sub
```

Le operazioni che è necessario eseguire sul nastro di carta sono sostanzialmente le seguenti:

- inserire un nuovo elemento;
- eliminare tutto il contenuto del nastro di carta;
- aggiornare il nastro;
- eliminare tutto il contenuto dell'array di stringhe;
- comporre una stringa unica dal concatenamento di tutte le stringhe contenute nell'array per salvare i dati in un file di testo;
- salvare la stringa in un file di testo;
- estrarre le singole stringhe da inserire nelle varie posizioni dell'array, durante la lettura di un file di testo contenente una "strisciata" salvata;
- stampare il nastro di carta.

Inserire un nuovo elemento

Molti pulsanti dell'interfaccia grafica inseriscono un nuovo elemento nel nastro di carta: è il caso dei pulsanti delle operazioni (+, -, *, /) e del pulsante di risultato (=).

Il metodo `Sub Inserisci` riceve come parametri il valore inserito nel display e il simbolo dell'operatore matematico corrispondente al pulsante premuto dall'utente.

Se il simbolo è uno tra le quattro operazioni matematiche, la lunghezza dell'array viene aumentata di una unità e vengono inseriti in tale posizione il parametro del valore, concatenato al simbolo dell'operatore. Subito dopo viene svuotato il nastro (`SvuotaNastro`) e nuovamente riempito con l'array aggiornato (`AggiornaNastro`).

```
Public Sub Inserisci(ByVal display As String, ByVal operatore As String)
    If operatore.Equals("+") Or operatore.Equals("-") Or _
       operatore.Equals("*") Or operatore.Equals("/") Then
        ReDim Preserve arrayNastro(arrayNastro.GetUpperBound(0) + 1)
        arrayNastro(arrayNastro.GetUpperBound(0)) = display & " " &
operatore
        Call SvuojaNastro()
        Call AggiornaNastro()
    End If
End Sub
```

Nel caso, invece, dell'operatore di eguaglianza (=), vengono inserite progressivamente tre righe nel nastro di carta, tra le quali una riga di trattini per separare gli operandi dal risultato, una riga di totale contrassegnata dal simbolo T e una riga di doppi trattini per sottolineare il risultato e dividerlo da un'eventuale operazione successiva.

```
ElseIf operatore.Equals("=") Then
    ReDim Preserve arrayNastro(arrayNastro.GetUpperBound(0) + 1)
    arrayNastro(arrayNastro.GetUpperBound(0)) = _
        "-----="
    ReDim Preserve arrayNastro(arrayNastro.GetUpperBound(0) + 1)
    arrayNastro(arrayNastro.GetUpperBound(0)) = display & " T"
    ReDim Preserve arrayNastro(arrayNastro.GetUpperBound(0) + 1)
    arrayNastro(arrayNastro.GetUpperBound(0)) = _
        "=====
```

```
End If
End Sub
```

Eliminare tutto il contenuto del nastro di carta

L'operazione svolta dal metodo `SvuotaNastro` è piuttosto semplice e serve a eliminare tutto il contenuto del nastro di carta.

Consiste solamente in un ciclo `Do While... Loop` nel quale il test è dato dal conteggio degli elementi presenti nella *ListBox* della form. Se il numero di elementi è maggiore di zero, viene eliminato il primo elemento della *ListBox* (l'elemento zero) e poi viene continuato il ciclo, eliminando a ogni iterazione l'elemento zero, fino al momento in cui la *ListBox* non contiene più alcun elemento.

```
Public Sub SvuojaNastro()
    Do While ctrl.Items.Count() > 0
        ctrl.Items.RemoveAt(0)
    Loop
End Sub
```

Aggiornare il nastro

L'operazione di aggiornamento del nastro di carta consiste nell'allineamento del contenuto del nastro stesso con gli elementi presenti nell'array.

Prima di tutto viene verificato se l'array contiene almeno un elemento in più dell'elemento che viene mantenuto vuoto (l'elemento di posizione zero). Se l'array ha un solo elemento, l'array è convenzionalmente vuoto e quindi non c'è nient'altro da fare.

```
Public Sub AggiornaNastro()
    If arrayNastro.GetUpperBound(0) = 0 Then
        ' non ci sono elementi da aggiungere al nastro
```

Se invece c'è almeno un elemento valido, viene avviato un ciclo di lettura di tutti gli elementi dell'array. Di ogni elemento viene estratta la stringa corrispondente e inserita in una distinta riga della *ListBox* del form.

```
Else
    For i As Integer = 1 To arrayNastro.GetUpperBound(0)
        ctrl.Items.Add(arrayNastro(i))
    Next
End If
```

Infine viene selezionato l'ultimo elemento della *ListBox* per realizzare l'effetto visivo dello scrolling, cioè dello spostamento del nastro di carta, quando il numero di elementi è superiore al numero di righe visibili nel controllo.

```
ctrl.SelectedIndex = ctrl.Items.Count - 1
End Sub
```

Eliminare tutto il contenuto dell'array

Questo è il metodo più semplice di tutti e consiste solamente nel ridimensionamento dell'array a una lunghezza di un elemento (l'elemento zero che viene mantenuto vuoto). Questo ridimensionamento viene effettuato per indicare che il nastro di carta deve essere svuotato perché non ci sono elementi da visualizzare.

```
Public Sub AzzeraNastro()
    ReDim Preserve arrayNastro(0)
End Sub
```

Comporre una stringa dal concatenamento delle stringhe dell'array

Il metodo `Function componiNastro` ha la funzione di comporre una stringa unica dal concatenamento di tutte le stringhe contenute nell'array per salvare i dati in un file di testo.

Prima di tutto viene definita una variabile locale (`risposta`) che viene utilizzata come variabile di appoggio per il concatenamento delle stringhe e contenente il valore da restituire al chiamante.

Subito dopo viene effettuato un test sul numero di elementi contenuti nell'array, come già è stato visto con il metodo `AggiornaNastro`. Nel caso in cui non ci siano elementi, la stringa restituita è semplicemente una stringa nulla.

```
Public Function componiNastro() As String
    Dim risposta As String = ""

    If arrayNastro.GetUpperBound(0) = 0 Then
        ' non ci sono elementi da aggiungere al nastro
```

Se invece esiste almeno un elemento valido, si procede alla scansione di tutti gli elementi validi contenuti nell'array, concatenando ciascuno nella variabile `risposta`.

Dopo che un elemento è stato concatenato, si verifica che nell'array esistano altri elementi da leggere: se la risposta è affermativa, viene concatenato anche un ritorno a capo (`Environment.NewLine`), per separare ciascuna riga dalle altre.

Infine viene restituita la stringa composta da tutte le righe lette nell'array.

```
Else
    For i As Integer = 1 To arrayNastro.GetUpperBound(0)
        risposta &= arrayNastro(i)
        If i < arrayNastro.GetUpperBound(0) Then
            risposta &= Environment.NewLine
        End If
    Next
End If
Return risposta
End Function
```

Salvare in un file di testo

Questo metodo riceve un parametro stringa che rappresenta un nome di file completo di percorso.

Prima di tutto viene dichiarata una variabile stringa di appoggio, alla quale viene assegnato il risultato del metodo `Function componiNastro`.

```
Public Sub SalvaNastro(ByVal nomeFile As String)
    Dim testo As String = componiNastro()
```

Se la stringa non è uguale a stringa di lunghezza zero, viene chiamato il metodo `WriteAllText`, come indicato in seguito, per salvare il contenuto della variabile in un file di testo.

```
If Not (testo.Equals("")) Then
    My.Computer.FileSystem.WriteAllText(nomeFile, testo, False)
```

```
End If
End Sub
```

Ricaricare un file di testo nell'array

Il metodo `Sub RecuperaNastro` viene utilizzato per leggere il contenuto di un file di testo salvato su disco e, da questo, estrarre le singole righe da inserire nelle varie posizioni dell'array. Tale metodo riceve un parametro di tipo stringa che rappresenta il nome del file di testo che si desidera recuperare. La lettura viene eseguita mediante il metodo `ReadAllText`, come si può vedere qui di seguito, alla cui stringa viene preventivamente concatenata una stringa di lunghezza nulla e un elemento di ritorno a capo. Questa modalità di caricamento viene effettuata per mantenere vuoto il primo elemento dell'array (l'elemento zero), così come è stato spiegato in precedenza.

```
Public Sub RecuperaNastro(ByVal nomeFile as String)
    Dim testo As String = "" & Environment.NewLine _
        & My.Computer.FileSystem.ReadAllText(nomeFile)
```

Le seguenti due istruzioni creano una istanza della classe `Regex`, alla quale viene passato come parametro un elemento di ritorno a capo che costituisce l'elemento separatore che dovrà utilizzare per separare le varie righe del testo caricato.

Nella seconda istruzione viene scomposto il testo caricato in singole righe e ciascuna di queste viene inserita nel successivo elemento dell'array, ricostruendo così la corretta rappresentazione del nastro di carta salvato.

```
Dim expreg As New Regex(Environment.NewLine)
arrayNastro = Regex.Split(testo, Environment.NewLine)
```

Le semplici chiamate ai metodi `SvuotaNastro` e `AggiornaNastro` sono già stati esaminati nel metodo `Inserisci`.

```
Call SvuotaNastro()
Call AggiornaNastro()
End Sub
```

Stampare il nastro di carta

Questo metodo non realizza la stampa effettiva del nastro di carta, ma si limita ad assegnare la stringa restituita dal metodo `componiNastro` nella *Label* di appoggio `TestoDaStampare` presente nel form. L'anteprima di stampa e la stampa vera e propria saranno poi eseguite con una finestra comune di dialogo richiamata direttamente dal form.

```
Public Sub StampaNastro()
    Calcolatrice.TestoDaStampare.Text = componiNastro()
End Sub
End Class
```

Il codice del form Calcolatrice.vb

Dopo aver esaminato attentamente la classe `clsNastro` è possibile fare la stessa operazione con il codice del form.

La prima parte, come al solito, contiene l'intestazione della classe form e le dichiarazioni delle variabili. Le variabili hanno le seguenti finalità:

- `flagPuntoDecimale` è un indicatore che rileva se il punto decimale è già stato inserito nel valore corrente, per impedire che possa essere inserito un secondo punto decimale;
- `cNastro` è un riferimento all'istanza di classe `clsNastro`;
- `Valore` contiene il valore memorizzato durante i calcoli;
- `documento` è un riferimento ad un oggetto di tipo `PrintDocument`, per realizzare l'anteprima di stampa.

```
Public Class Calcolatrice
    Private flagPuntoDecimale As Boolean = False
    Private cNastro As clsNastro
    Private Valore As Double = 0
    ' dichiara un oggetto PrintDocument di nome documento (per anteprima)
    Private WithEvents documento _
        As New System.Drawing.Printing.PrintDocument
```

A seguire, viene gestito l'evento `Load` del form `Calcolatrice` che viene scatenato al caricamento del form stesso. Nel gestore di questo evento vengono effettuate due operazioni: la creazione dell'istanza di classe `clsNastro`, passandogli il riferimento alla `ListBox`, e la chiamata al metodo `InizializzaPrintPreviewDialog` che sarà analizzato in seguito.

```
Private Sub Calcolatrice_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    cNastro = New clsNastro(Me.Nastro)
    InizializzaPrintPreviewDialog()
End Sub
```

A questo punto viene riportato il codice di gestione dell'evento `Click` dei dieci pulsanti numerici (da 0 a 9) che semplicemente chiamano il metodo `premiTastoNumerico`, passando un carattere che indica il tasto premuto.

```
Private Sub Tasto1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Tasto1.Click
    premiTastoNumerico("1")
End Sub
```

```
Private Sub Tasto2_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Tasto2.Click
    premiTastoNumerico("2")
End Sub
```

```
Private Sub Tasto3_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Tasto3.Click
    premiTastoNumerico("3")
End Sub
```

```
Private Sub Tasto4_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Tasto4.Click
    premiTastoNumerico("4")
End Sub
```

```
Private Sub Tasto5_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Tasto5.Click
    premiTastoNumerico("5")
End Sub
```

```
Private Sub Tasto6_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Tasto6.Click
```

```

        premiTastoNumerico("6")
    End Sub

    Private Sub Tasto7_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Tasto7.Click
        premiTastoNumerico("7")
    End Sub

    Private Sub Tasto8_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Tasto8.Click
        premiTastoNumerico("8")
    End Sub

    Private Sub Tasto9_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Tasto9.Click
        premiTastoNumerico("9")
    End Sub

    Private Sub Tasto0_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Tasto0.Click
        premiTastoNumerico("0")
    End Sub

```

Ecco quindi il codice del metodo `premiTastoNumerico`. Questo metodo accoda la corrispondente cifra, ricevuta come parametro, sposta nuovamente il focus sul display e posiziona il cursore nel punto più a destra dell'ultima cifra inserita:

```

    Private Sub premiTastoNumerico(ByVal tastoNumerico As String)
        Display.Text = Display.Text & tastoNumerico
        Me.Display.Focus()
        Me.Display.SelectionStart = Me.Display.Text.Length
    End Sub

```

Un frammento di codice interessante è quello riportato nel seguito. Il gestore dell'evento `KeyPress` del display permette di verificare, mentre il focus è impostato sul display, se il tasto premuto sia uno di quelli relativi a cifre numeriche. Nel caso la verifica fosse positiva, sarà accodata la cifra al valore già presente nel display.

Questo metodo realizza una funzione estremamente utile: infatti permette di inserire velocemente dei numeri utilizzando il tastierino della tastiera fisica, piuttosto che obbligare l'utente a fare clic sui pulsanti numerici del tastierino presente sul form.

Si noti che il metodo chiamato nel caso di tasti numerici è lo stesso presente nei gestori degli eventi click dei corrispondenti pulsanti da zero a nove.

In questo stesso gestore dell'evento `KeyPress` sono state codificate le azioni da eseguire nel caso di altri tasti premuti: i tasti delle operazioni (+, -, *, /), il punto decimale, il tasto di eguaglianza (=) e perfino il tasto *Invio* che viene parificato all'inserimento di un segno di eguaglianza. Questo significa che per stampare il totale dell'operazione è possibile premere il tasto "=" così come è sufficiente premere *Invio*.

Infine, l'istruzione `Case Else` permette di escludere qualsiasi altro tasto premuto, per evitare che nel display venga fatto il tentativo di inserire caratteri non ammessi, come caratteri alfabetici, simboli non corretti e così via. Infatti l'istruzione `e.KeyChar = ""` non fa altro che "buttare via" il carattere che è stato premuto, per evitare che il gestore della pressione dei tasti di default, che viene comunque chiamato subito dopo il termine di questo codice, tenti di interpretare un tasto non ammesso.

```

Private Sub Display_KeyPress(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.KeyPressEventArgs) _
    Handles Display.KeyPress
    Dim tast0 As New Control
    Select Case e.KeyChar
        Case "1", "2", "3", "4", "5", "6", "7", "8", "9", "0"
            premiTastoNumerico(e.KeyChar)
        Case "."
            Call Me.PuntoDecimale_Click(sender, e)
        Case "+"
            Call Me.TastoPiu_Click(sender, e)
        Case "-"
            Call Me.TastoMeno_Click(sender, e)
        Case "*"
            Call Me.TastoPer_Click(sender, e)
        Case "/"
            Call Me.TastoDiviso_Click(sender, e)
        Case "="
            Call Me.TastoUguale_Click(sender, e)
        Case Environment.NewLine
            Call Me.TastoUguale_Click(sender, e)
        Case Else
            e.KeyChar = ""
    End Select
    Me.Display.Focus()
    Me.Display.SelectionStart = Me.Display.Text.Length
End Sub

```

Nel tastierino numerico sono presenti altri due elementi che si riferiscono a elementi numerici che devono essere aggiunti al numero digitato e pertanto anche questo codice viene riportato qui di seguito. Il primo accoda tre zeri (per indicare le migliaia, per esempio):

```

Private Sub Tasto00_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Tasto00.Click
    Display.Text = Display.Text & "000"
    Me.Display.Focus()
    Me.Display.SelectionStart = Me.Display.Text.Length
End Sub

```

Il secondo metodo accoda il simbolo del punto decimale. Si noti che il punto decimale viene accodato solo se l'indicatore `flagPuntoDecimale` non è già attivato (`True`). Nel caso tale indicatore fosse ancora impostato a `False`, viene aggiunto il punto decimale e impostato a `True`, per impedire l'inserimento di più punti decimali nello stesso valore.

```

Private Sub PuntoDecimale_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles PuntoDecimale.Click
    If flagPuntoDecimale = False Then
        Display.Text = Display.Text & "."
        flagPuntoDecimale = True
    End If
    Me.Display.Focus()
    Me.Display.SelectionStart = Me.Display.Text.Length
End Sub

```

Altri due gestori di eventi `Click` permettono, rispettivamente, di eseguire le operazioni relative alla pressione dei pulsanti `C` e `CE`:

- il pulsante `C` permette di cancellare il contenuto del display e di annullare l'indicatore di punto decimale. Il contenuto della variabile `Valore` e il contenuto del nastro di carta non vengono

modificati;

- il pulsante *CE* permette invece di cancellare il contenuto del display, della variabile *Valore* e del nastro di carta. In pratica la calcolatrice viene reimpostata come se fosse appena stata avviata.

```
Private Sub TastoC_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TastoC.Click
    Display.Text = ""
    flagPuntoDecimale = False
    Me.Display.Focus()
End Sub

Private Sub TastoCE_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TastoCE.Click
    Call cNastro.AzzeraNastro()
    Call cNastro.SvuotaNastro()
    Call cNastro.AggiornaNastro()
    Display.Text = ""
    flagPuntoDecimale = False
    Valore = 0
    Me.Display.Focus()
    Me.Display.SelectionStart = Me.Display.Text.Length
End Sub
```

Il gestore dell'evento `Click` del pulsante `TastoCancellaUno` (il pulsante presente sulla destra del display) verifica se il carattere meno significativo della stringa presente nel display è un punto decimale. In caso positivo, viene reimpostato a `False` il corrispondente indicatore. In ogni caso viene eliminato il carattere più a destra e aggiornato il display.

```
Private Sub TastoCancellaUno_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TastoCancellaUno.Click
    If Display.Text.Substring(Display.Text.Length - 1, 1) = "." Then
        flagPuntoDecimale = False
    End If
    Display.Text = Display.Text.Substring(0, Display.Text.Length - 1)
    Me.Display.Focus()
    Me.Display.SelectionStart = Me.Display.Text.Length
End Sub
```

Nel caso vengano premuti i pulsanti relativi alle operazioni matematiche (+, -, *, /), viene effettuata una chiamata al metodo `eseguiOperazione`, al quale viene passato il simbolo come parametro:

```
Private Sub TastoPiu_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TastoPiu.Click
    Call eseguiOperazione("+")
End Sub

Private Sub TastoMeno_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TastoMeno.Click
    Call eseguiOperazione("-")
End Sub

Private Sub TastoPer_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TastoPer.Click
    Call eseguiOperazione("*")
End Sub
```

```

Private Sub TastoDiviso_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TastoDiviso.Click
    Call eseguiOperazione("/")
End Sub

```

Viene quindi riportato il codice del metodo `eseguiOperazione`, dove è possibile notare che nella prima parte vengono eseguite le operazioni matematiche specifiche del simbolo ricevuto come parametro e viene inserito nel nastro di carta il valore e il simbolo corrispondente, mentre nella seconda parte viene svuotato il display, reimpostato a `False` l'indicatore del punto decimale e spostato il focus sul display:

```

Private Sub eseguiOperazione(ByVal operatore As String)
    Select Case operatore
        Case "+"
            Valore += CType(Display.Text, Double)
            Call cNastro.Inserisci(Display.Text, "+")

        Case "-"
            Valore -= CType(Display.Text, Double)
            Call cNastro.Inserisci(Display.Text, "-")

        Case "*"
            Valore *= CType(Display.Text, Double)
            Call cNastro.Inserisci(Display.Text, "*")

        Case "/"
            Valore /= CType(Display.Text, Double)
            Call cNastro.Inserisci(Display.Text, "/")

    End Select
    Display.Text = ""
    flagPuntoDecimale = False
    Me.Display.Focus()
End Sub

```

L'ultimo gestore di evento `Click` che interessa il tastierino numerico del form è quello relativo al tasto di eguaglianza (`TastoUguale`). Il codice che segue:

- chiama il metodo `Inserisci` dell'istanza inerente al nastro di carta;
- aggiunge il risultato anche nel display, permettendo quindi il suo riutilizzo, per esempio in una somma successiva;
- azzera il `Valore`, reimposta a `False` l'indicatore del punto decimale;
- aggiorna il nastro di carta, dopo averne cancellato il contenuto;
- infine sposta il focus sul display, posizionando il cursore a destra della stringa:

```

Private Sub TastoUguale_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TastoUguale.Click
    Call cNastro.Inserisci(Valore.ToString, "=")
    Display.Text = Valore
    Valore = 0
    flagPuntoDecimale = False
    Call cNastro.SvuotaNastro()
    Call cNastro.AggiornaNastro()
    Me.Display.Focus()
    Me.Display.SelectionStart = Me.Display.Text.Length
End Sub

```

Le funzionalità di salvataggio su file e di recupero del contenuto del nastro di carta da un file di testo sono piuttosto interessanti, perché permettono di memorizzare permanentemente anche lunghe sequenze di operazioni per poterle verificare in seguito o magari per inviarle via posta elettronica a un collega.

Il salvataggio su file di testo è realizzato con il gestore dell'evento `Click` del pulsante `TastoSalva`, riportato qui di seguito. Una volta impostate le proprietà del controllo `SaveFileDialog1`, viene mostrata la finestra di dialogo per ottenere il nome del file su cui salvare, dopo di che viene semplicemente chiamato il metodo `SalvaNastro` della classe `clsNastro`:

```
Private Sub TastoSalva_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TastoSalva.Click
    Dim nomeFile As String = ""
    SaveFileDialog1.InitialDirectory = "c:\"
    SaveFileDialog1.FileName = "C:\prova.txt"
    SaveFileDialog1.Filter = "file di testo (*.txt)|*.txt"
    SaveFileDialog1.FilterIndex = 1
    SaveFileDialog1.RestoreDirectory = True
    If SaveFileDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
        nomeFile = Me.SaveFileDialog1.FileName
        cNastro.SalvaNastro(nomeFile)
    End If
    Me.Display.Focus()
    Me.Display.SelectionStart = Me.Display.Text.Length
End Sub
```

Allo stesso modo il gestore dell'evento `Click` del pulsante `TastoRecupera`, imposta le proprietà del controllo `OpenFileDialog1`, apre la relativa finestra di dialogo per la scelta del file da ricaricare e chiama il metodo `RecuperaNastro` della classe `clsNastro`:

```
Private Sub TastoRecupera_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TastoRecupera.Click
    OpenFileDialog1.InitialDirectory = "c:\"
    OpenFileDialog1.FileName = "C:\prova.txt"
    OpenFileDialog1.Filter = "file di testo (*.txt)|*.txt"
    OpenFileDialog1.FilterIndex = 1
    OpenFileDialog1.RestoreDirectory = True

    If OpenFileDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
        Dim nomeFile As String = Me.OpenFileDialog1.FileName
        cNastro.RecuperaNastro(nomeFile)
    End If
    Me.Display.Focus()
    Me.Display.SelectionStart = Me.Display.Text.Length
End Sub
```

Per le funzionalità di anteprima di stampa è stato utilizzato un codice molto simile a quello mostrato nell'esempio del capitolo 9, nel paragrafo che ha trattato il controllo `PrintPreviewDialog`.

Il metodo `InizializzaPrintPreviewDialog` e i gestori degli eventi `Click` del pulsante `TastoAnteprima` e `PrintPage` dell'istanza documento, sono quindi stati adattati con poche modifiche.

```
' inizializza il controllo
Private Sub InizializzaPrintPreviewDialog()
```

```

' imposta dimensione, posizione e nome
Me.PrintPreviewDialog1.ClientSize = _
    New System.Drawing.Size(400, 500)
Me.PrintPreviewDialog1.Location = New System.Drawing.Point(29, 29)
Me.PrintPreviewDialog1.Name = "PrintPreviewDialog1"

' imposta la dimensione minima della finestra di dialogo
Me.PrintPreviewDialog1.MinimumSize = _
    New System.Drawing.Size(375, 250)

' imposta la proprietà UseAntiAlias a True per permettere
' al sistema operativo di utilizzare l'effetto antialiasing
Me.PrintPreviewDialog1.UseAntiAlias = True
End Sub

Private Sub TastoAnteprima_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles TastoAnteprima.Click

    Me.TestoDaStampare.Text = cNastro.componiNastro

    ' imposta la proprietà PrintPreviewDialog.Document all'oggetto
    ' PrintDocument selezionato dall'utente
    PrintPreviewDialog1.Document = documento

    ' chiama il metodo ShowDialog che scatenerà l'evento
    ' PrintPage del documento
    PrintPreviewDialog1.ShowDialog()

    Me.Display.Focus()
    Me.Display.SelectionStart = Me.Display.Text.Length
End Sub

```

La modifica più consistente è quella presente appunto nel gestore dell'evento `PrintPage` dell'istanza documento. Infatti il testo da stampare non è, in questo caso, una stringa fissa e preimpostata, ma è il risultato ottenuto dal metodo `Function formattaStampa`, al quale è stato passato il contenuto della *Label* nascosta presente sul form:

```

Private Sub documento_PrintPage(ByVal sender As Object, _
    ByVal e As System.Drawing.Printing.PrintPageEventArgs) _
    Handles documento.PrintPage

    ' Qui va inserito il codice per disegnare la pagina.
    ' Questo codice sarà chiamato da PrintPreviewDialog.Show

    ' Il seguente codice prepara un semplice messaggio
    ' sul documento creato nella finestra di dialogo
    Dim text As String = formattaStampa(Me.TestoDaStampare.Text)
    Dim printFont As New System.Drawing.Font _
        ("Courier New", 14, System.Drawing.FontStyle.Regular)
    e.Graphics.DrawString(text, printFont, _
        System.Drawing.Brushes.Black, 500, 100)
End Sub

```

Infine viene riportato proprio il metodo `Function formattaStampa` che si accolla l'onere di:

- prendere il testo ricevuto come parametro e distribuirne le singole righe in un array, con la stessa tecnica delle espressioni regolari già presentata;
- formattare ogni singolo elemento dell'array, aggiungendo degli spazi a sinistra, per raggiungere una lunghezza fissa complessiva di 22 caratteri;

- restituire nuovamente il testo completo, componendolo mediante accodamento di ogni singola riga con un elemento di ritorno a capo.

```

Private Function formattaStampa(ByVal testoNastro As String) As String
    Dim expreg As New Regex(Environment.NewLine)
    Dim arrayStampa() As String = _
        Regex.Split(testoNastro, Environment.NewLine)
    Dim riga As String = ""
    Dim spazi As String = "                " ' 22 spazi
    Dim i As Integer

    For i = 0 To arrayStampa.GetUpperBound(0)
        riga = arrayStampa(i)
        riga = spazi & riga
        riga = riga.Substring(riga.Length - 22, 22)
        arrayStampa(i) = riga
    Next

    testoNastro = arrayStampa(0)
    For i = 1 To arrayStampa.GetUpperBound(0)
        testoNastro &= Environment.NewLine & arrayStampa(i)
    Next
    Return testoNastro
End Function
End Class

```

La calcolatrice funzionante è quella mostrata in Figura 12.2.

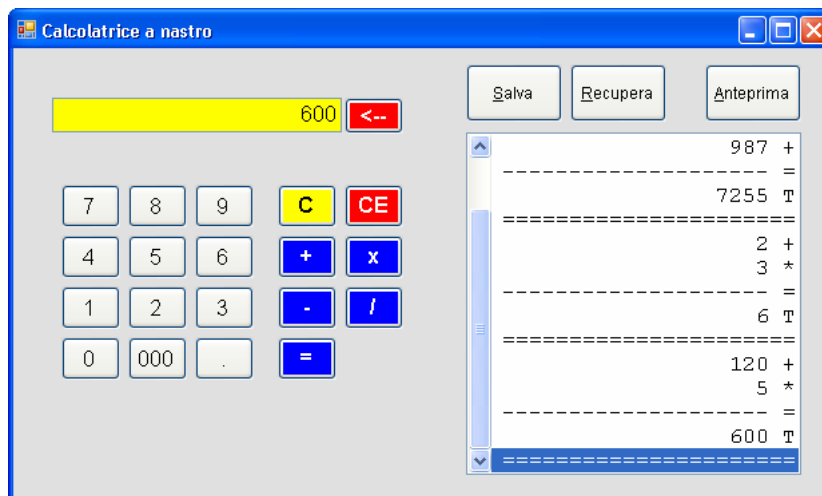


Figura 12.2 La calcolatrice a nastro funzionante.

In Figura 12.3, invece, è presentata la finestra di anteprima di stampa, con zoom pari al 100%. In tale finestra è possibile modificare il fattore di zoom, scorrere le pagine e impostare la visualizzazione su più pagine (per documenti multipagina) e stampare sulla stampante di sistema predefinita.

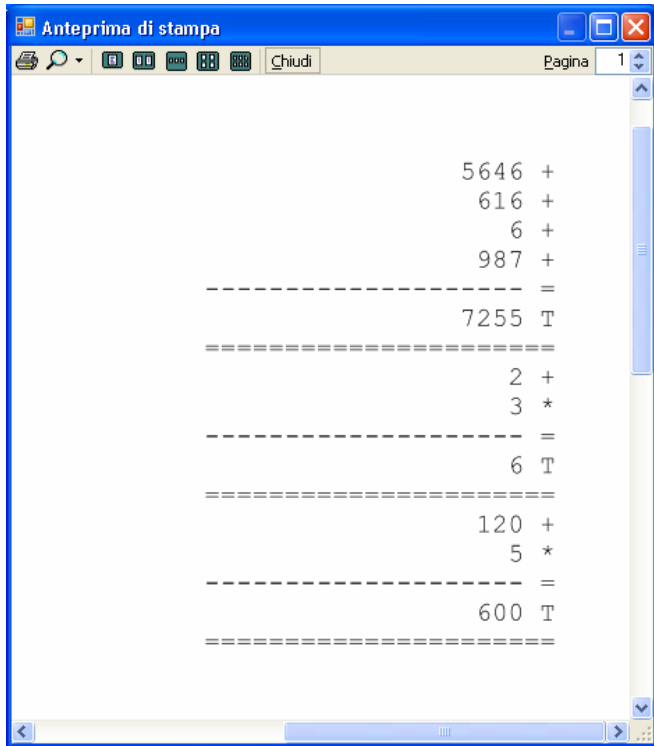


Figura 12.3 La finestra di anteprima di stampa.

NOTA DELL'AUTORE

Il presente capitolo è un capitolo inedito, in quanto escluso dalla pubblicazione per motivi di spazio. Avrebbe dovuto essere pubblicato come ultimo capitolo del libro i cui riferimenti sono riportati nel seguito della presente pagina.

Sul sito "**Visual Basic Tips & Tricks**" (www.visual-basic.it) e sul sito dell'autore (www.deghetto.it) **saranno pubblicati anche tutti gli esempi del libro**, compreso l'esempio di questo capitolo inedito. Gli esempi sono forniti in sorgente e in eseguibile. Per utilizzare gli esempi è necessario avere almeno una copia di Visual Basic 2005 Express.

Chi dovesse trovare errori o imprecisioni di qualunque genere è invitato gentilmente a comunicarlo a mezzo e-mail (mario@deghetto.it), sul forum "DOTNET" di "Visual Basic Tips & Tricks" (www.visual-basic.it) oppure sul blog di Mario De Ghetto (<http://community.visual-basic.it>).

"VISUAL BASIC 2005 e il Framework .NET 2.0"

Autore: Mario De Ghetto

Numero 15 - Bimestrale, maggio 2006 - Distribuzione Parrini
Collana "**Guide Pratiche**" di **Computer Magazine**
Codice 9771825062009

Editore:

Future Media Italy

Via Asiago, 45 - 20128 Milano
Tel. 02-252916.1 - fax 02-26005121

Arretrati:

tel. 02-252916209 - fax 02-26005512
dal lunedì al venerdì: 9.00/12.30 - 13.30/17.30
e-mail arretrati@futureitaly.it

Distribuzione per l'edicola:

Parrini & C. SpA
Viale Forlanini 23 - 20134 Milano
Via Vitorchiano 81 - 00189 Roma