PROGRAMMAZIONE
con

Microsoft Visual Basic
Express Edition
2008

Prof.ssa Maria Teresa Tattoli
Prof.ssa Lucia Tattoli

Indice

Il Framework .NET 3
Visual Basic.NET 3
Visual Basic Express Edition 4
IDE di VB 2008 Express Edition 5
Controlli Windows Form 11
Il codice 14
Il Form Designer 16
Esecuzione del programma e debugging 16
Salvataggio di un progetto 19
Le variabili e i tipi di dati 22
Le costanti 24
Istruzioni di assegnazione 24
Operatori aritmetici 24
Operatori sulle stringhe 25
Le strutture della programmazione 26
La struttura di selezione: If … Then … Else 26
ElseIf e Select Case 28
Le strutture iterative 30
Finestre di dialogo predefinite di VB 2008: InputBox 32
Finestre di dialogo predefinite di VB 2008: MsgBox 33
I principali controlli di VB 2008 36
Gruppo Controlli comuni 36
Gruppo Contenitori 41
Gruppo menu e barre degli strumenti 43
Gruppo Dati 51
Variabili semplici e strutturate 56
Gli array 56
I vettori 56
Le matrici 61
I record 64
I Sottoprogrammi 66
Le funzioni 67
Istruzioni Exit Function e Exit Sub 69
Visibilità di variabili e sottoprogrammi 69
Funzioni predefinite: librerie del Visual Basic 71
Funzioni matematiche 71
Funzioni per la gestione delle stringhe 71
Funzioni per la conversione di tipo 73
La gestione delle date 74
I File 75
I file ad accesso sequenziale 75
I file ad accesso diretto (random) 77
Progetti multiform 80
Aggiungere un form ad un progetto 80
Metodi ShowDialog e Show 81
Metodo Hide 81
Istruzione Close 81
Eventi Activated, Deactivate, Load e Unload del form 81
Le finestre di dialogo (Common Dialog Box) 82
Le finestre di dialogo Apri e Salva 83
La finestra di dialogo Colore 86
La finestra di dialogo Carattere 87
Compilare e distribuire un'applicazione 88
Creare un pacchetto d'installazione 89
Installazione del programma 95

Il Framework .NET

Nella produzione del software, il **framework** è una struttura di supporto su cui un software può essere organizzato e progettato. Alla base di un framework c'è una serie di librerie di codice utilizzabili con uno o più linguaggi di programmazione, spesso corredate da una serie di strumenti di supporto allo sviluppo del software (IDE, debugger, ecc.)
Lo scopo di un framework è di risparmiare allo sviluppatore la riscrittura di codice già realizzato in precedenza per compiti simili. Questa esigenza si è presentata sempre più spesso man mano che le interfacce utente sono diventate più complesse, o, più in generale, man mano che è aumentata la produzione di software con funzionalità secondarie simili. Ad esempio, il tipo di interazione con l'utente offerta da un menu a tendina sarà sempre la stessa, indipendentemente dall'applicazione cui il menu appartiene; in casi come questo, un framework, che permette di aggiungere la funzionalità di una finestra con un menu a tendina con poche righe di codice sorgente, permetterà al programmatore di concentrarsi sulle funzionalità specifiche dell'applicazione, senza doversi far carico di scrivere codice "di contorno". Il termine inglese framework quindi può essere tradotto come **intelaiatura** o **struttura**, che è appunto la sua funzione, a sottolineare il fatto che al programmatore rimane solo da creare il contenuto vero e proprio dell'applicazione.
Per far girare il software basato su un certo framework è necessario installarlo nell'ambito del sistema operativo.

Windows .NET Framework è il framework che Microsoft ha creato come piattaforma di sviluppo software per applicazioni desktop, server e web. La sua caratteristica peculiare è di essere indipendente dalla versione di Windows su cui è installata e di includere molte funzionalità progettate espressamente per integrarsi in ambiente internet e garantire il massimo grado di sicurezza e integrità dei dati.
L'architettura del .NET Framework rende inoltre particolarmente agevole la programmazione "cross language", cioè con differenti linguaggi; in particolare, Microsoft ne supporta due: C# e VB.NET, entrambi completamente **orientati agli oggetti**, il primo ispirato ad una sintassi "C like" ma del tutto somigliante a Java, il secondo evoluzione del predecessore Visual Basic. Tuttavia esistono tanti altri linguaggi con cui si può programmare sulla piattaforma .NET; ad esempio, Delphi.NET, distribuito da Borland. I linguaggi supportabili dal Framework .NET sono oltre venti, alcuni dei quali integrati nel **Visual Studio .NET** di Microsoft, un ambiente di sviluppo integrato sviluppato da Microsoft, che supporta diversi tipi di linguaggi, quali C++, C#, J#, Visual Basic.Net e ASP.Net, e che permette la realizzazione di applicazioni, siti web, applicazioni web e servizi web.
Le **versioni** finora realizzate del Windows .NET Framework sono:

- .NET Framework 1.0 (2002)
- .NET Framework 1.1 (2004)
- .NET Framework 2.0 (2005)
- .NET Framework 3.0 (2006)
- .NET Framework 3.5 (2007)
- .NET Framework 4.0 (2010)

In genere, il .NET Framework si inserisce nel mercato paragonandosi a Java ma, mentre quest'ultimo ha lo scopo principale di fornire un unico linguaggio per tutte le piattaforme e i sistemi operativi esistenti, .NET persegue lo scopo di fornire il supporto a più linguaggi ma basati su un'unica piattaforma, il .NET Framework appunto, tuttora distribuito con i sistemi Windows e con i dispositivi mobili SmartPhone e PocketPC.

Il progetto **Mono** prevede la portabilità di .Net su sistemi operativi non Windows. Già attualmente un applicativo compilato con il .Net framework può funzionare sotto altri sistemi (per esempio Linux) con l'installazione del framework Mono.

Visual Basic.NET

Con l'uscita del .NET Framework, Microsoft ha introdotto una nuova "visione" del suo più famoso linguaggio di sviluppo, denominata Visual Basic.NET. Non si tratta di un aggiornamento del vecchio VB6, ma di un vero e proprio nuovo linguaggio che integra tutte le classi di base del Framework .NET, mantenendo una sintassi simile a quella di VB6.

Questo cambiamento ha interrotto la compatibilità verso il basso di Visual Basic, ma ha colmato tutte le carenze tipiche del VB6. Sono state introdotte tutta una serie di funzionalità tipiche del .NET Framework. Anche se esiste un tool per la migrazione delle applicazioni, alcune funzionalità non possono essere convertite automaticamente, richiedendo quindi uno sforzo successivo per completare la migrazione.
Molte delle originali carenze sono state colmate, ma questo ha reso Visual Basic .Net un linguaggio più complesso del suo predecessore, al tempo stesso, però, estremente più potente e versatile.

Le versioni di Visual Basic .NET finora realizzate, in concomitanza con le principali versioni del Framework .NET, sono:

- **Visual Basic .NET 7.0** o semplicemente **Visual Basic .NET**, nato nel 2002, è stato implementato sul .NET Framework 1.0. Benché commercialmente sia il successore del VB6, in realtà l'unico punto in comune è la sintassi VB like.
- **Visual Basic 8.0** o **Visual Basic 2005**, nato nel 2005, è stato implementato sul .NET Framework 2.0. Non si usa più la keyword .NET nel nome, in quanto da questo momento sarà sottointeso che Visual Basic è basato sul .NET Framework.
- **Visual Basic 9.0** o **Visual Basic 2008**, nato nel 2008, è stato implementato sul .NET Framework 3.5. Con questa versione del linguaggio, è possibile realizzare web con la tecnologia Silverlight.
- **Visual Basic 10.0** o **Visual Basic 2010**, nato nel 2010, è stato implementato sul .NET Framework 4.0.

Ogni versione introduce nuove funzionalità e caratteristiche sintattiche.

Anche in Visual Basic .NET, come nelle precedenti versioni di Visual Basic, la programmazione è visuale, basata sulla tecnologia **Windows Forms**. Con programmazione visuale si intende l'attività di sviluppo del software basato sull'uso dell'interfaccia grafica (finestre, pulsanti, icone) tramite la manipolazione grafica degli elementi e non tramite sintassi scritta.

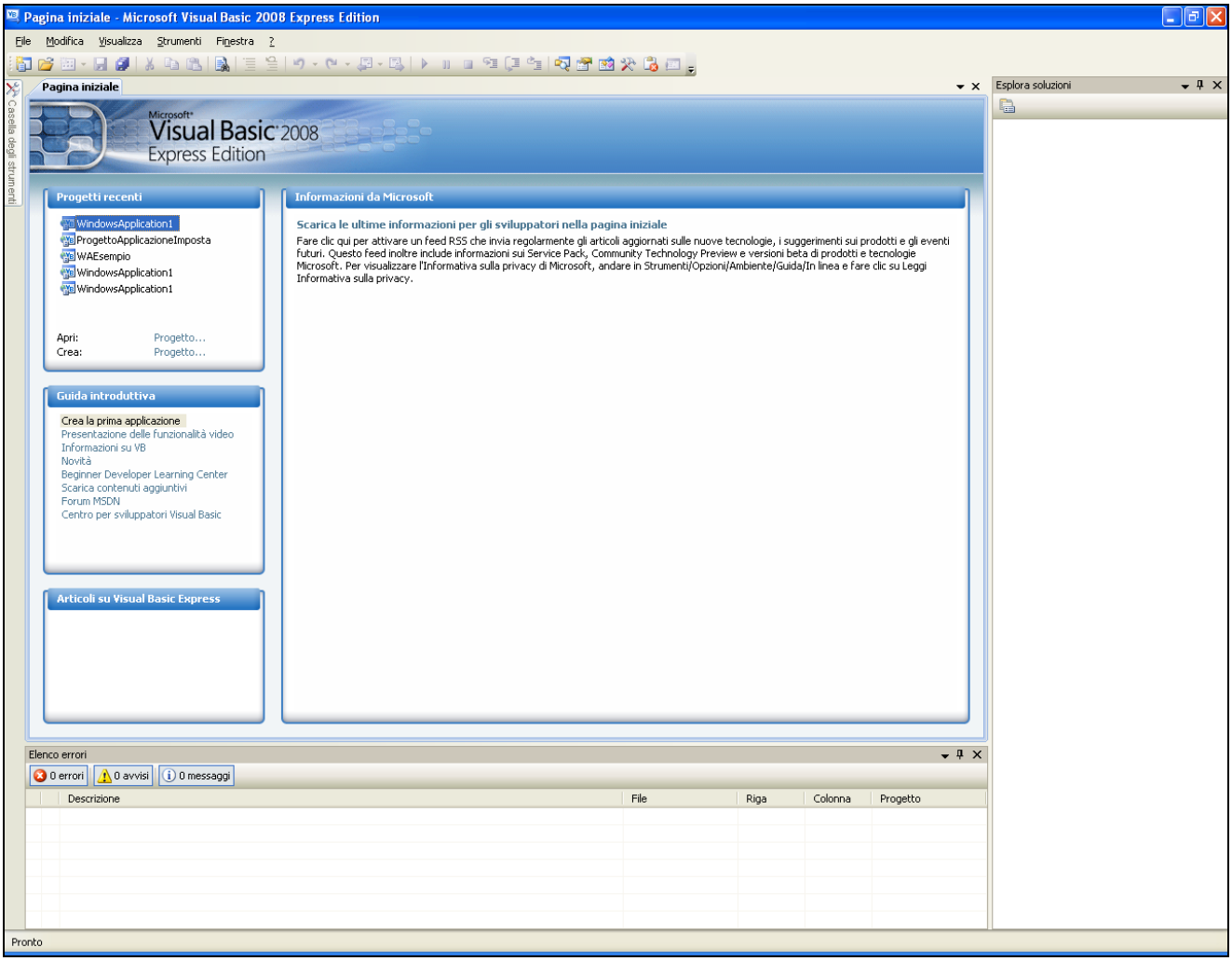
Visual Basic Express Edition

Visual Basic Express Edition è una versione limitata di Visual Basic.NET. Può essere utilizzata sia come uno strumento per imparare a programmare in Visual Basic, sia come uno strumento di sviluppo completo di ogni funzionalità per programmatori che non hanno necessità di utilizzare la versione completa di Visual Basic.
Le versioni Express Edition di Visual Basic.NET sono del tutto gratuite e possono essere scaricate dal sito di Microsoft. Il link per la versione 2008 è:

http://www.microsoft.com/express/download/

IDE di VB 2008 Express Edition

Quando si apre Visual Basic Express Edition per la prima volta, la finestra **Pagina iniziale** (**Start Page**) occupa la maggior parte dell'**IDE** (**I**ntegrated **D**evelopment **E**nvironment = Ambiente di sviluppo integrato. È l'ambiente di lavoro).

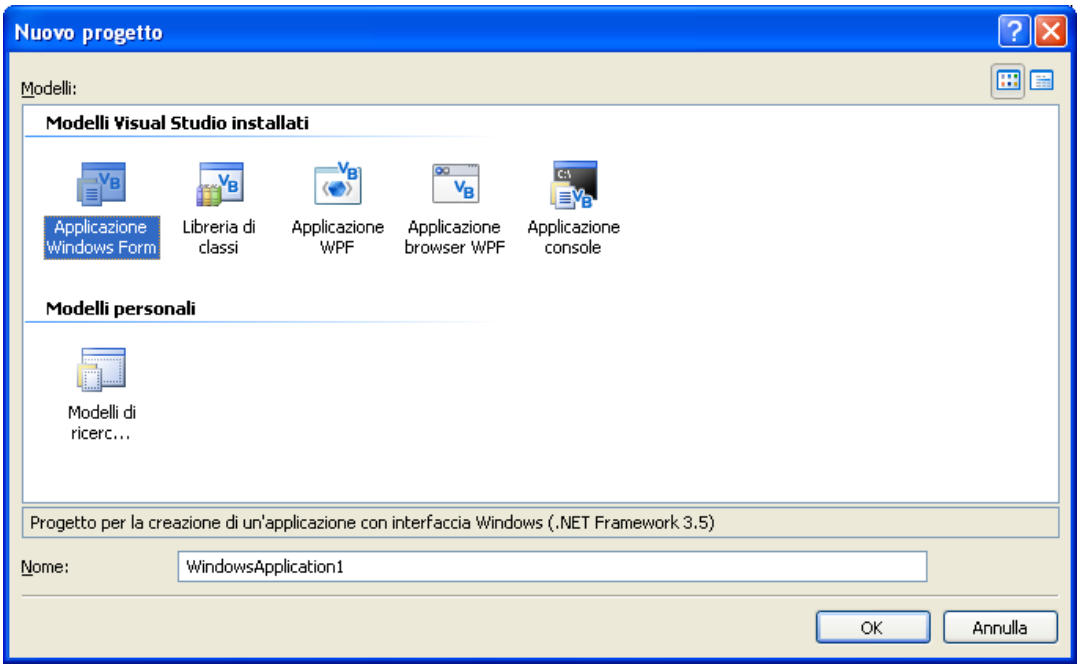


Essa si presenta composta dai seguenti riquadri:

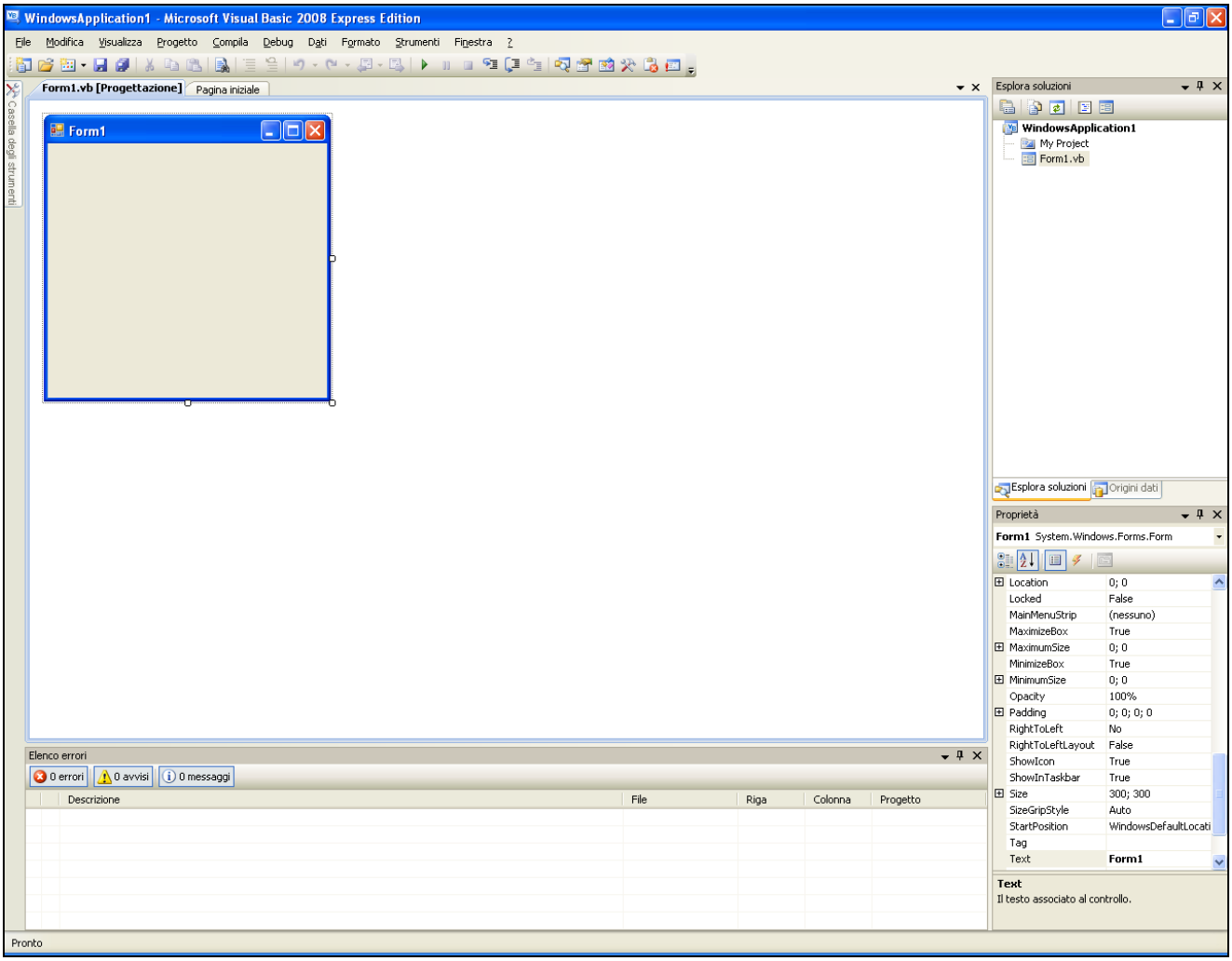
- **Progetti recenti** (*Recent Project*). Sono i progetti su cui si è lavorato più di recente, che possono essere aperti da questo riquadro in modo rapido.
- **Guida introduttiva** (*Getting Started*). In questo riquadro vengono raccolti alcuni collegamenti a manuali disponibili sia in locale che su siti internet predefiniti.
- **Articoli su Visual Basic Express** (*Visual Basic Headlines*). Se si è connessi a Internet, tale elenco viene aggiornato a intervalli regolari.
- **Informazioni da Microsoft** (*Visual Basic Developer News*). Anche questa sezione si può aggiornare dinamicamente, mediante la connessione ad internet.

Nel lato destro dell'IDE è presente la **finestra Esplora soluzioni**. Questa finestra, inizialmente vuota, è una delle più utilizzate di Visual Basic 2008, perché consente all'utente di gestire i file delle cosiddette **Soluzioni**. Una soluzione è l'insieme di uno o più progetti che vengono sviluppati in modo integrato.

Per creare un nuovo progetto, si seleziona **File** – **Nuovo progetto**.
Appare la finestra seguente, da cui si deve selezionare l'opzione **Applicazione Windows Forms**.



A questo punto viene creata una nuova soluzione contenente il progetto e la finestra di VB 2008 assume il seguente aspetto:



È questa la **Modalità di progettazione**.

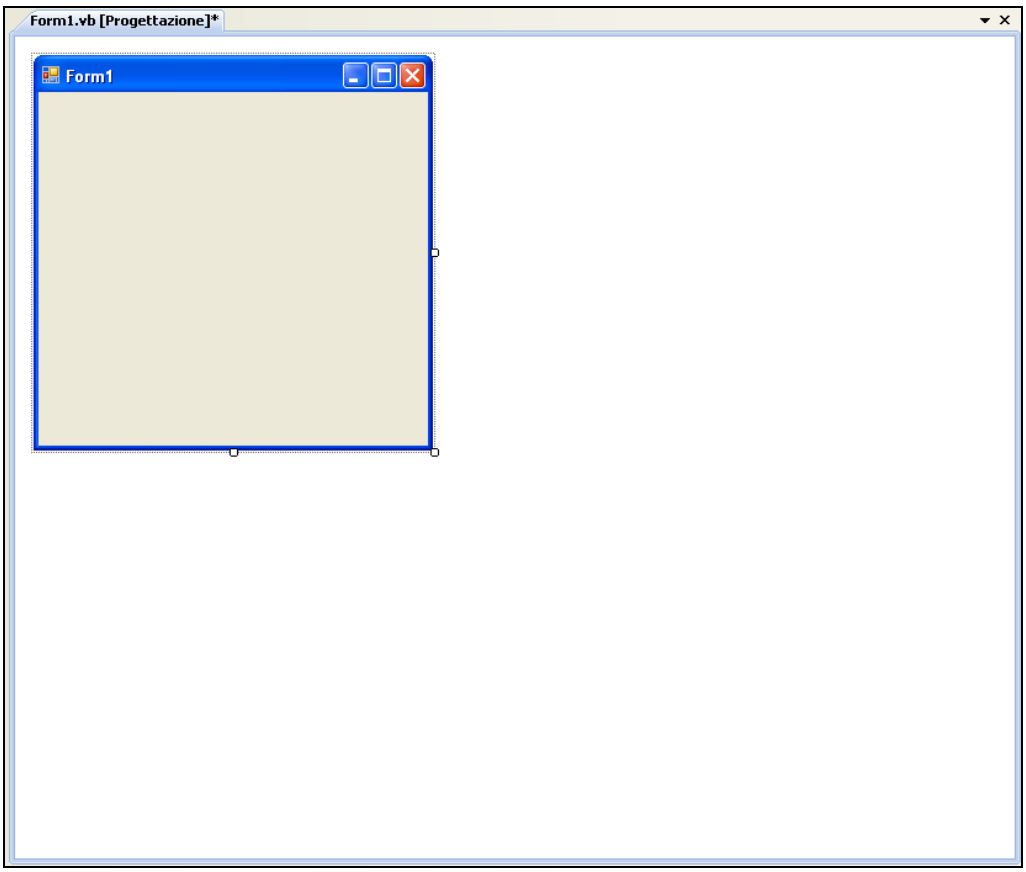
Per aggiungere un altro progetto ad una soluzione, selezionare **File** – **Aggiungi** (**Add**) – **Nuovo Progetto** (**New Project**) oppure **Progetto Esistente** (**Existing Project**).

Cliccando su una delle voci della finestra Esplora soluzioni, nella sottostante **Finestra delle Proprietà** (**Properties**) appaiono le proprietà ad essa riferite, che possono essere modificate. In questa finestra è possibile impostare anche le proprietà di tutti i componenti dell'interfaccia utente.

Nella parte superiore dell'IDE sono presenti una **barra dei menu** e una **barra degli strumenti**. I menu e i pulsanti della barra degli strumenti variano in base all'attività corrente.

Nella parte inferiore dell'IDE è presente una **barra di stato** in cui è visualizzata la dicitura **Pronto**. Durante l'utilizzo dell'IDE, nella barra di stato vengono visualizzati messaggi correlati all'attività corrente, ad esempio informazioni sullo stato di avanzamento di un progetto.

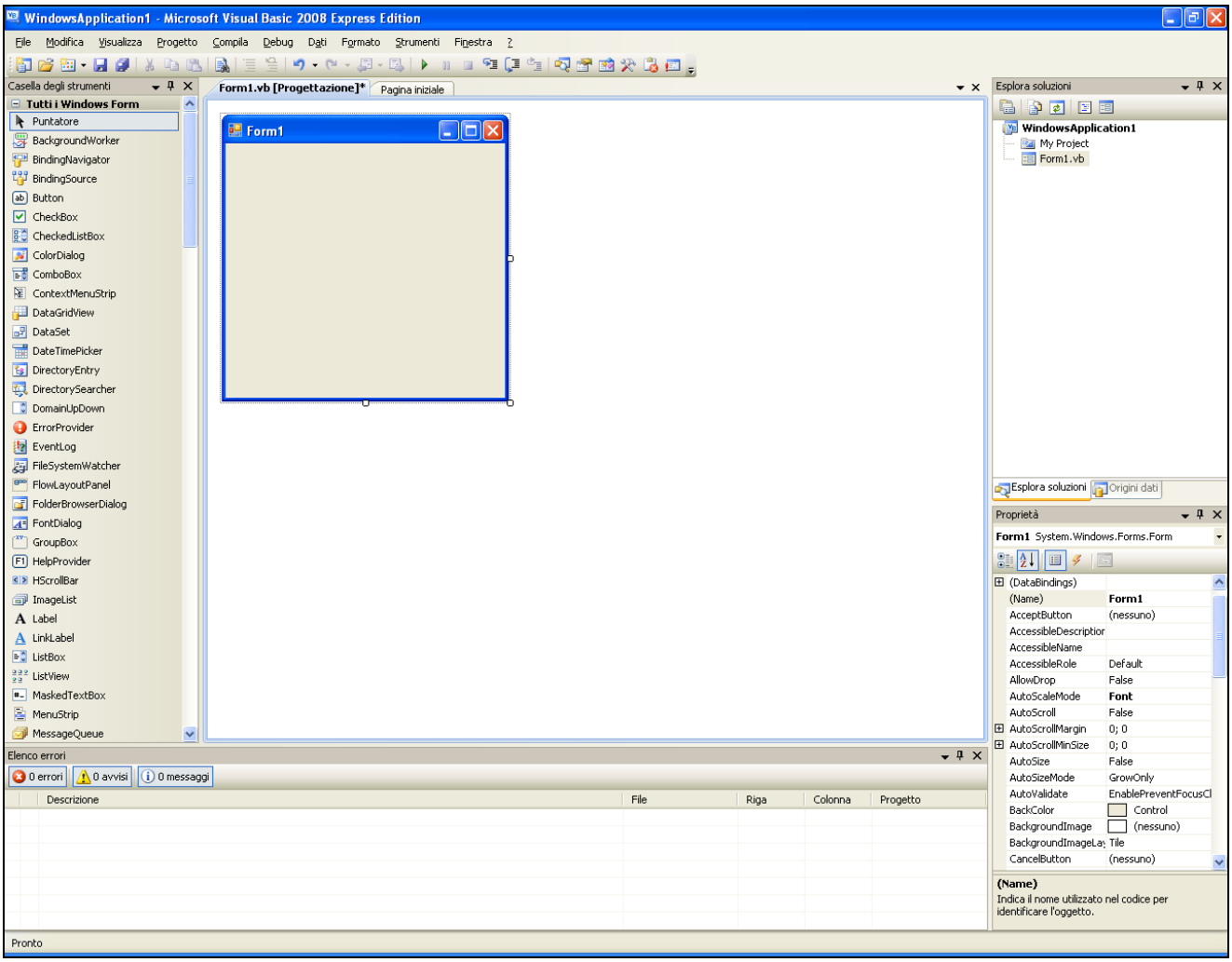
In modalità progettazione, la Pagina iniziale viene coperta da un'altra finestra denominata **Finestra di progettazione del Form**.



Il Form rappresenta la finestra che sarà visualizzata quando l'utente eseguirà l'applicazione; è inizialmente vuoto, ma su di esso possono essere collocati in modo ordinato tutti gli **oggetti** che serviranno all'utente per eseguire il programma. Tali oggetti, che prendono il nome di **controlli**, sono raccolti, rappresentati da icone, nella cosiddetta **Casella degli strumenti** (**Toolbox**), una finestra che si trova normalmente sul lato sinistro dell'IDE.

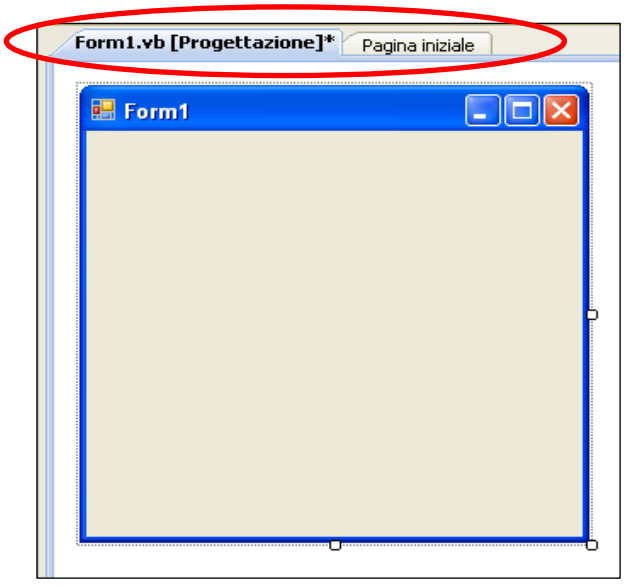
La casella degli strumenti è un riquadro che contiene tutti i controlli che possono essere posizionati sul form per costruire l'interfaccia. Essa può essere costantemente visibile oppure può essere resa visibile temporaneamente e in sovraimpressione sul form, premendo il pulsante bianco con su scritto "Casella degli strumenti", che si trova sul lato sinistro dell'IDE. Per passare da uno stato all'altro, si può agire sul piccolo pulsante a forma di puntina da disegno che si trova nella barra del titolo della casella degli strumenti (è il secondo dei tre pulsanti a destra).

La puntina da disegno appare in verticale se la Toolbox è costantemente visibile, in orizzontale se la Toolbox è nascosta e deve essere visualizzata premendo sull'apposito pulsante. Lo stesso meccanismo vale anche per le altre finestre di utilità dell'IDE (Proprietà, Esplora soluzioni, ecc.)

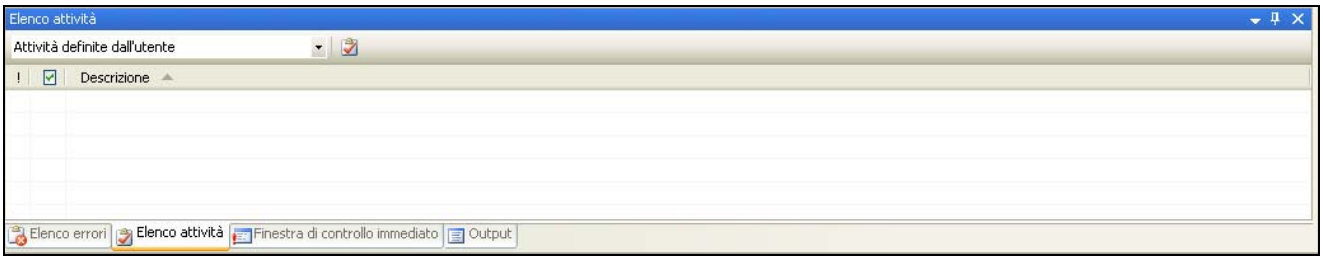


È possibile accedere di nuovo alla pagina iniziale selezionando la scheda corrispondente sopra la finestra di progettazione del Form. Durante le attività di progettazione potranno essere aperte anche altre finestre, a ciascuna delle quali corrisponderà una scheda.

Le finestre con schede contrassegnate da un **asterisco** (*****) hanno contenuti che non sono stati salvati.



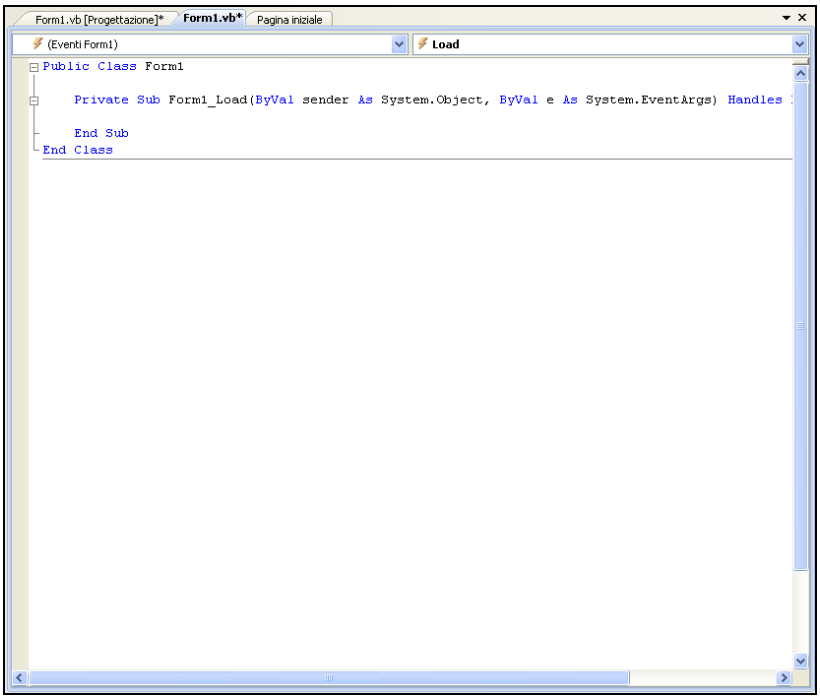
Nella parte bassa dell'IDE vi è uno spazio riservato alla visualizzazione di alcune finestre di servizio:



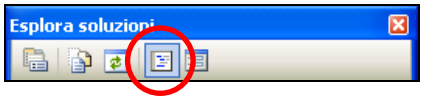
- La finestra **Elenco attività** (*Task list*), nella quale è possibile immettere note sulle operazioni da eseguire e riferimenti a specifici punti del codice. Tali punti sono individuati da commenti, ovvero frasi descrittive che iniziano con un apostrofo; se, dopo l'apostrofo, si scrive un token, che può essere **TODO** (da fare), **HACK** (da modificare), **UNDONE** (incompleto), il commento viene automaticamente inserito nell'elenco attività, dove, cliccando due volte su un commento, si arriva al punto del codice da esso individuato. Se non è visibile, si può visualizzare la finestra **Elenco attività** selezionando, dal menu principale, **Visualizza – Altre finestre – Elenco attività**.
- La finestra **Elenco errori**, che riporta gli errori effettuati in fase di progettazione. Fare doppio clic su un messaggio di errore per aprire e visualizzare il punto in cui si è verificato l'errore. Se non è visibile, si può visualizzare la finestra **Elenco errori** selezionando, dal menu principale, **Visualizza – Elenco errori**.
- Finestre di dialogo aggiuntive per il controllo dei valori delle variabili, la visualizzazione dell'output e l'esecuzione di altre attività di debug possono essere visualizzate in fase di esecuzione selezionandole dal menu **Debug**.

Se sono aperte più finestre, si può passare dall'una all'altra attraverso le relative schede che si trovano nella parte sottostante.

Se si fa doppio clic su un form o un controllo, viene visualizzato l'**editor di codice**, ovvero una nuova finestra nella quale è possibile scrivere le istruzioni in linguaggio per l'applicazione.

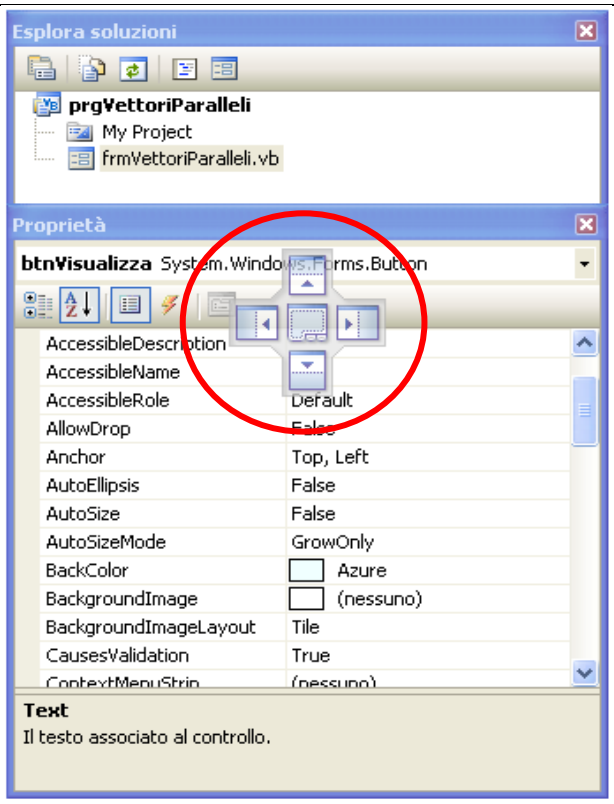
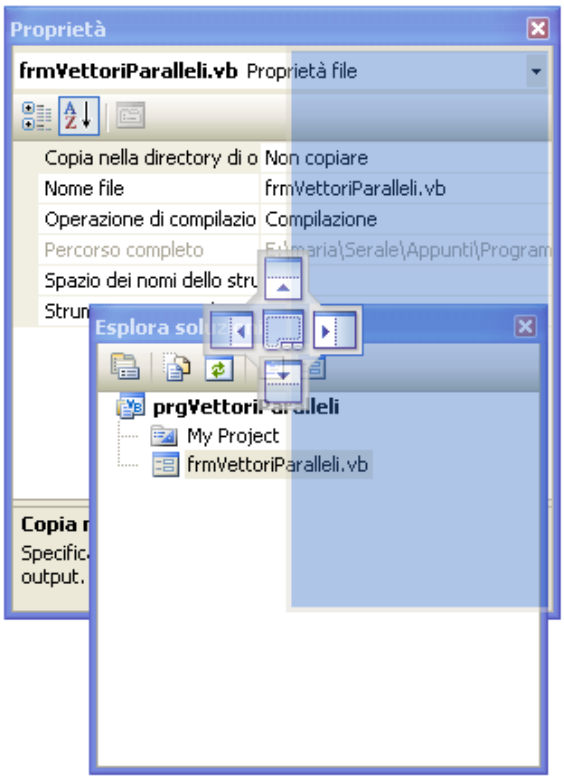
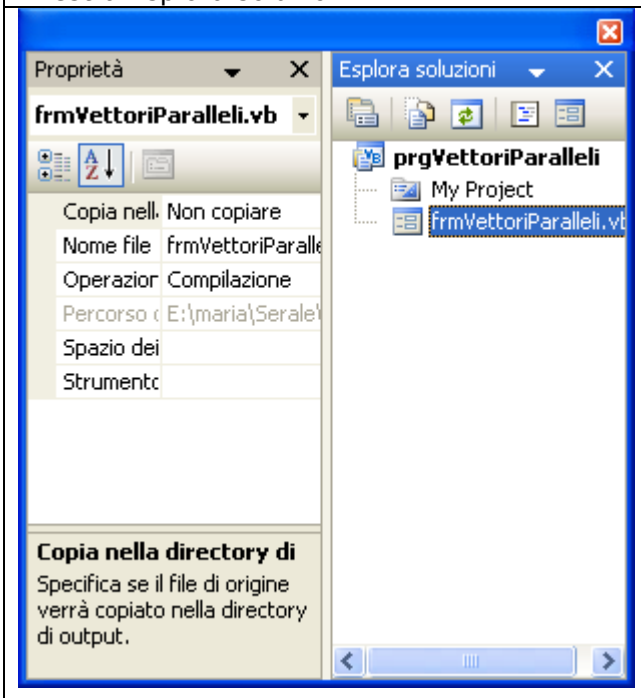
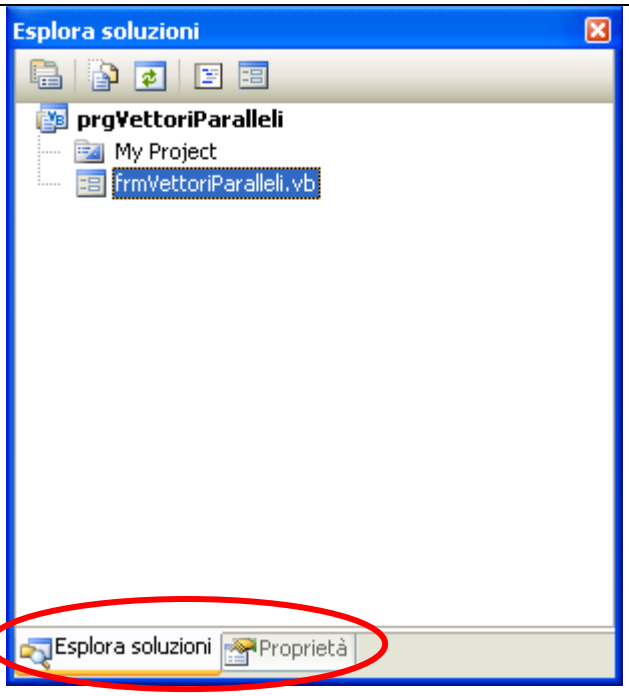


La finestra del codice appare anche cliccando sul pulsante **Visualizza codice** della finestra **Esplora soluzioni**:



Visual Basic Express Edition consente di personalizzare l'IDE modificando il layout delle finestre, scegliendo le finestre da visualizzare, aggiungendo o rimuovendo comandi o pulsanti della barra degli strumenti e altro.

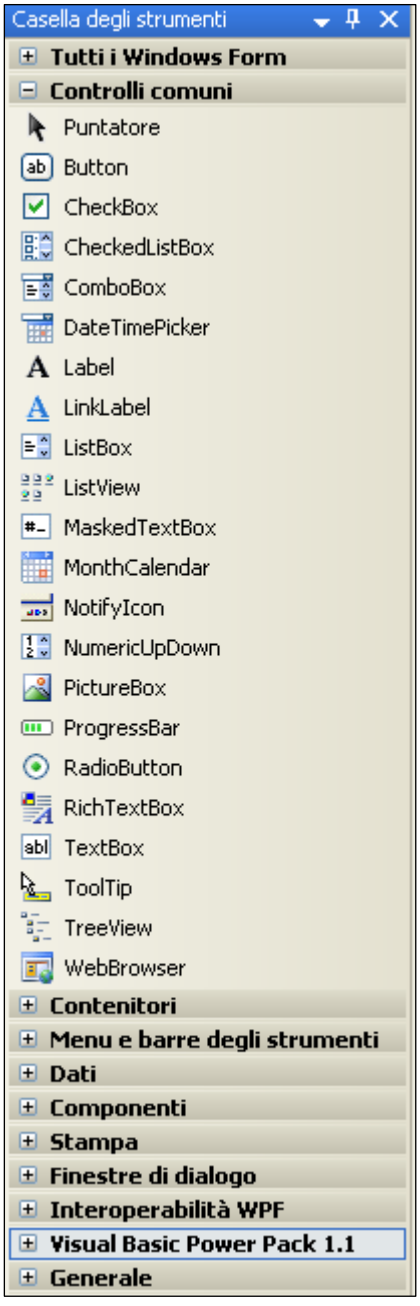
Due o più finestre dell'ambiente di lavoro possono anche occupare lo stesso riquadro. Basta trascinare (si agisce con il mouse sulla barra del titolo) una finestra sull'altra. Compare, in trasparenza, un indicatore di posizione composto da cinque piccole icone, delle quali ne deve essere selezionata una spostando la finestra in modo da evidenziarla (compare una fascia blu).
Le quattro icone sul contorno permettono di affiancare la finestra che si sta spostando a destra, a sinistra, sopra o sotto rispetto a quella su cui è avvenuto lo spostamento. L'icona al centro permette invece di spostare la finestra nello stesso riquadro di quella su cui è sovrapposta, facendo comparire dei pulsanti di tabulazione in basso, per passare dall'una all'altra.

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>Si sovrappone la finestra delle proprietà alla finestra Esplora soluzioni.</td>
<td>Si sceglie di affiancare la finestra a destra.</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>Queste sono le finestre affiancate.</td>
<td>In questo caso invece le finestre sono state sovrapposte e si può passare dall'una all'altra attraverso i pulsanti di tabulazione.</td>
</tr>
</table>

Controlli Windows Form

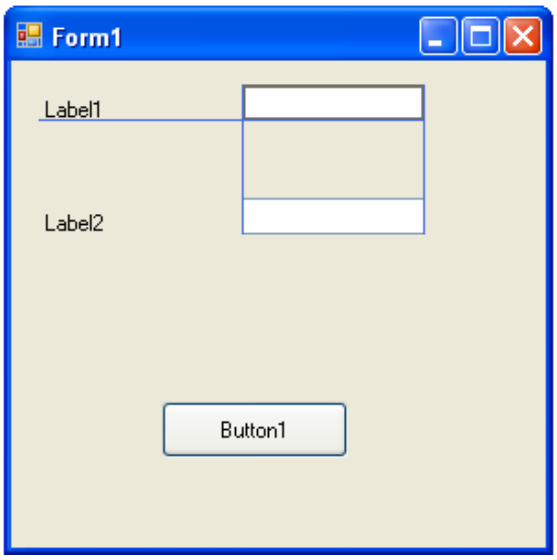
Dato che Visual Basic mette a disposizione una grande quantità di controlli, questi sono stati raggruppati nella Toolbox in categorie, in modo tale da agevolare la ricerca di un controllo specifico.
A sinistra del nome di ciascuna categoria, c'è un piccolo pulsante (+/-) che consente di espandere/comprimere l'elenco di controlli che vi appartengono.

I raggruppamenti predefiniti sono i seguenti:

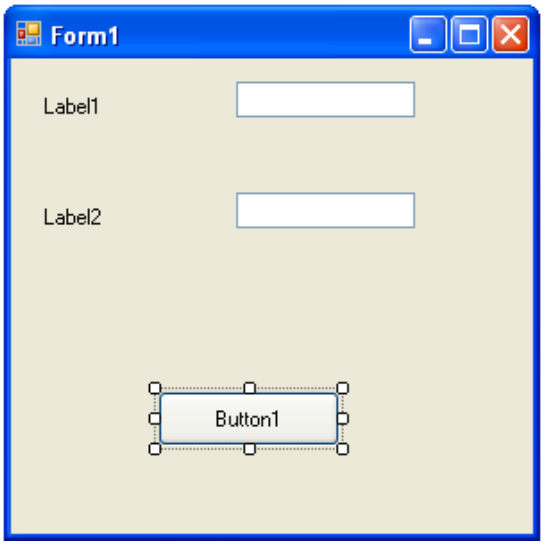


- **Tutti i Windows Form** (*All Windows Forms*) comprende tutti i controlli disponibili, disposti in ordine alfabetico, con l'aggiunta di un pulsante puntatore (pointer), che non è un vero e proprio controllo, ma permette di selezionare un controllo già presente sul from.
- **Controlli comuni** (*Common control*), che comprende tutti i controlli più comunemente utilizzati nelle applicazioni Windows, tra cui (v. anche pag. 33):
  - **Label**, un'etichetta che consente di visualizzare messaggi sul Form,
  - **TextBox**, una casella di testo che viene generalmente usata per consentire all'utente di inserire e/o modificare dati,
  - **Button**, un pulsante cliccando sul quale l'utente può impartire ordini.
- **Contenitori** (*Containers*), che comprende alcuni controlli che fungono da contenitori di altri controlli, come il pannello, il GroupBox (il frame di VB6), il TabControl.
- **Menu e barre degli strumenti** (*Menus & Toolbars*), che mette a disposizione i controlli necessari alla realizzazione di un menu tradizionale, di barre di pulsanti e barre di stato.
- **Dati** (*Data*), che comprende controlli visuali per la gestione dei dati, per esempio una vista di dati in forma tabellare.
- **Componenti** (*Components*), che contiene controlli che spaziano in molte tipologie di gestione del sistema, come il timer, il controllo dei processi, della porta seriale, ecc.
- **Stampa** (*Printing*), che comprende controlli in grado di gestire l'impostazione della stampante, l'anteprima di stampa e la stampa vera e propria di un documento.
- **Finestre di dialogo** (*Dialogs*), che include le classiche finestre di dialogo per la selezione dei colori, l'apertura o il salvataggio di un documento, la selezione di un font.
- **Interoperabilità WPF** (*WPF Interoperability*), un nuovo gruppo introdotto per utilizzare le nuove funzionalità WPF (Windows Presentation Foundation) del Framework 3.0. WPF è una tecnologia emergente per le interfacce grafiche.
- **Visual Basic Power Pack 1.1**, che contiene il controllo PrintForm, che consente di stampare un Windows Form come un report standard.
- **Generale** (*General*), inizialmente vuoto, ma vi si possono inserire nuovi controlli che chiunque può creare.

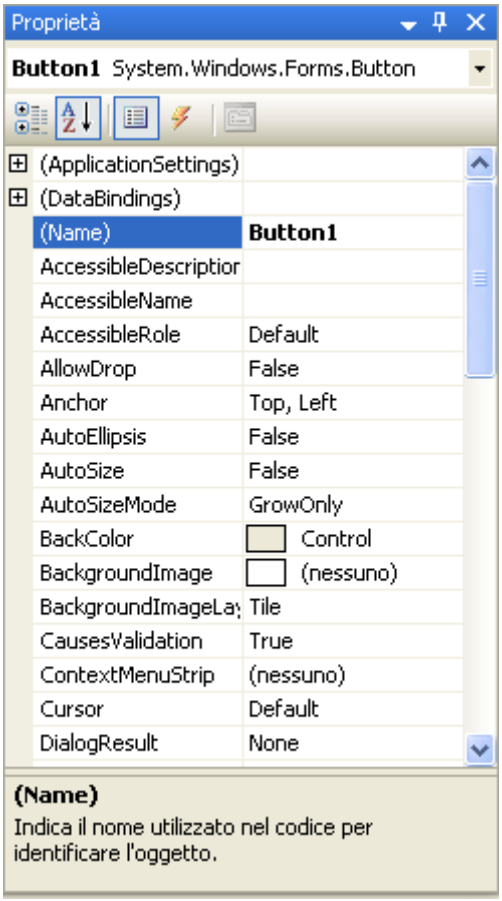
Per posizionare un controllo su un form, si può selezionare il controllo nella Toolbox e trascinarlo, con la tecnica del Drag & Drop sul form.
In alternativa, si può cliccare due volte sul controllo nella Toolbox. Se in quel momento sul form c'è un controllo selezionato, il nuovo controllo apparirà con uno spostamento di pochi pixel in basso a destra rispetto al controllo selezionato. Se invece nessun controllo è selezionato, il nuovo controllo apparirà nell'angolo in alto a sinistra del form.
Si può anche selezionare un oggetto nella ToolBox e disegnarlo sul form, trascinando il mouse.
È possibile poi spostare i controlli trascinandoli con il mouse. Una guida colorata permette di allinearli facilmente a quelli già presenti.



Per selezionare un oggetto (form o controllo), basta cliccare su di esso.
Quando un oggetto è selezionato, compaiono, nel suo contorno, le **maniglie**, che consentono di ridimensionarlo.



La **Finestra delle Proprietà** visualizza le proprietà dell'oggetto selezionato sul Form.
All'inizio, dato che il Form è vuoto, la finestra delle proprietà elencherà proprio le proprietà del Form; inserendo un oggetto qualunque sul Form, verranno visualizzate le proprietà dell'oggetto appena inserito. A seconda dell'oggetto selezionato, le proprietà visualizzate nella tabella saranno differenti.
Per modificare una proprietà, occorre fare clic sulla casella che la affianca, nella colonna a destra. A questo punto può essere richiesto di immettere manualmente un valore oppure scegliere tra più valori elencati o ancora potrà comparire una finestra di dialogo (ad esempio una **palette di colori**) collegata all'argomento.



Ogni controllo presente nella casella degli strumenti (ed anche il form) è dotato di un insieme di proprietà, metodi ed eventi.

Le **proprietà** rappresentano gli attributi che definiscono l'aspetto e le varie funzionalità di ogni controllo. Tra le proprietà più comuni figurano:

- la proprietà **Name**, che indica il nome con cui l'oggetto è identificato all'interno del codice;
- le proprietà **Height** e **Width** (unite nella proprietà **Size**), che indicano l'altezza e la larghezza dell'oggetto;
- la proprietà **AutoSize**, se impostata a True, fa in modo che il controllo si autoridimensioni in base al suo contenuto (Size non è in questo caso impostabile).
- le proprietà **BackColor** e **ForeColor**, che indicano rispettivamente il colore dello sfondo dell'oggetto e il colore dei caratteri scritti su di esso;

- la proprietà **Text**, per la scrittura di un titolo (ad esempio, per il Form) o di testo su un oggetto (ad esempio, per Label e CommandButton);
- la proprietà **TextAlign** per definire il tipo di allineamento del testo.
- la proprietà **Image** carica un'immagine nel controllo. La proprietà **ImageAlign** permette di definire il tipo di allineamento dell'immagine.
- la proprietà **Font**, per impostare il tipo di carattere, lo stile e la dimensione;
- la proprietà **Visible** rende visibile o invisibile il controllo a seconda che sia impostata a *True* o *False*;
- la proprietà **Enabled** abilita o disabilita il controllo a seconda che sia impostata a *True* o *False*.
- **Cursor**: seleziona la forma che deve avere il cursore quando si sovrappone al controllo.

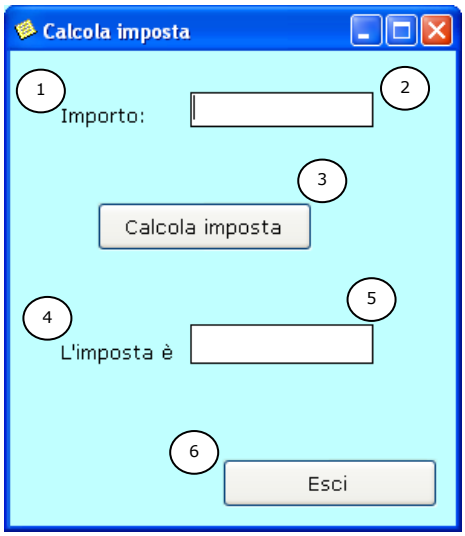
Tutti i controlli dispongono della proprietà **Name**.
Quando si creano nuovi controlli all'interno del form, i loro nomi vengono generati automaticamente tramite un indice, preceduto dal nome della classe a cui il controllo appartiene, come, ad esempio, Button1, Button2, ... per i pulsanti, TextBox1, TextBox2, ... per le caselle di testo, Label1, Label2, ... per le Label, e così via.
Se per applicazioni veloci, che devono svolgere poche, semplici operazioni, per le quali basta una finestra anche piccola, non c'è problema a lasciare i controlli nominati in questo modo, quasi sempre è utile, anzi, molto utile, rinominarli in modo che il loro scopo sia comprensibile anche da codice e che il loro nome sia molto più facile da ricordare. E in questo caso è necessario adottare uno stile ben preciso anche per i nomi. Uno dei criteri più usati prende il nome di **Notazione ungherese**: il nome è una stringa composta, per i primi tre caratteri, da una sigla che identifica il tipo di controllo e, per i rimanenti, da parole che ne descrivano la funzione. Ad esempio, una TextBox in cui digitare il cognome di una persona si potrebbe chiamare txtCognome, un pulsante che debba eseguire un'operazione si potrebbe chiamare btnEsegui, e così via.

I prefissi generalmente usati per i controlli più comuni sono i seguenti:

Controllo	Prefisso
Button	btn
TextBox	txt
Label	lbl
RadioButton	rdb
CheckBox	chk
ListBox	lst
ComboBox	cmb
PictureBox	pic

Si consideri, ad esempio, il form in figura, su cui sono stati posizionati 6 controlli:

	Tipo	Proprietà Name	Proprietà Text
-	Form	frmImposta	Calcola imposta
1	Label	lblImportoMsg	Importo
2	TextBox	txtImporto	
3	Button	btnCalcola	Calcola imposta
4	Label	lblImpostaMsg	L'imposta è
5	Label	lblImposta	
6	Button	btnCalcola	Calcola imposta



Si osservi il controllo n. 5: è una label usata per visualizzare un risultato (che ovviamente l'utente non deve poter modificare). La label è stata resa somigliante alla TextBox impostando opportunamente alcune proprietà:

- **Autosize** = *True*, per rendere la dimensione della Label indipendente dal contenuto; con la proprietà **Size** se ne possono poi impostare le dimensioni.
- **BackColor** = *White*, che è comunemente il colore delle TextBox.
- **BorderStyle** = *Fixed3D*, lo stesso assegnato alla TextBox, che determina lo stile del bordo.

Un **metodo** è un'azione che l'oggetto è in grado di compiere. Ad es., l'oggetto Form dispone dei metodi **Show** e **Hide**, il primo mostra il Form, il secondo lo nasconde. Come per le proprietà, alcuni metodi sono comuni a diversi controlli, altri sono specifici di determinati controlli.

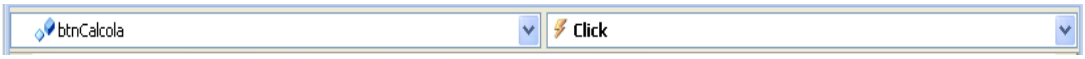
Gli **eventi** sono, come dice il nome, "situazioni" generate dal controllo quando si verificano certe condizioni; solitamente, ma non necessariamente, gli eventi si producono in conseguenza di un'azione dell'utente: per es., quando l'utente fa clic su un pulsante, il controllo che identifica quel pulsante genera un evento **Click**.
L'evento Click è tra i più comuni tra gli oggetti, come anche:

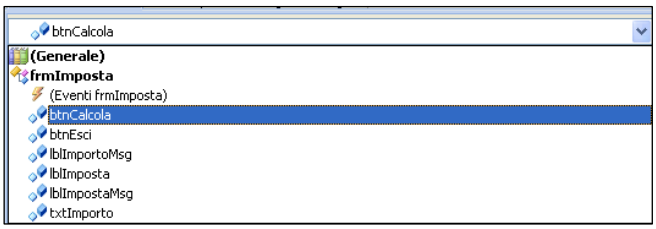
- **DoubleClick**, che si verifica quando l'utente clicca due volte sull'oggetto.
- **GotFocus**, che si verifica quando l'oggetto riceve il focus, ad esempio perché l'utente clicca su di esso.
- **LostFocus**, che si verifica quando l'oggetto perde il focus, ad esempio perché l'utente clicca su un altro oggetto.

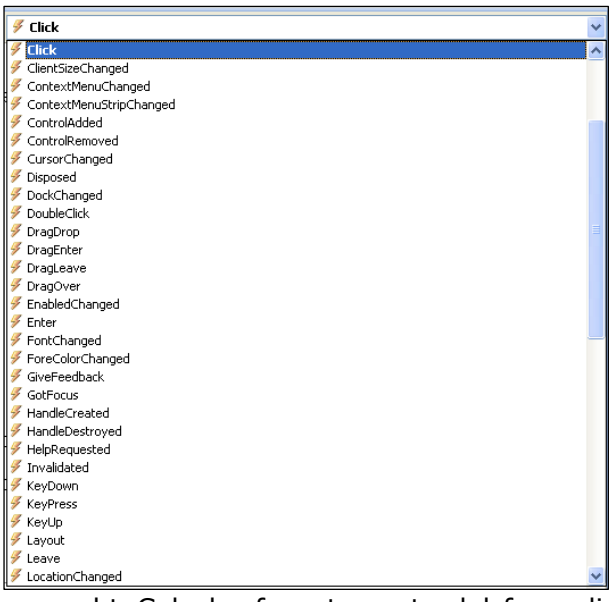
Una volta realizzata l'interfaccia utente, occorre programmare gli eventi a cui il programma deve rispondere, cioè scrivere le istruzioni che devono essere eseguite quando l'utente effettua un'azione significativa su uno degli oggetti dell'interfaccia. Per ogni coppia oggetto-evento individuata, verrà realizzata una **procedura**, vale a dire una sequenza di istruzioni che risponde all'evento.
Per il form precedente, ad esempio, può essere programmato l'evento click su ciascun pulsante, in modo che il programma esegua operazioni quando l'utente cliccherà con il mouse su di essi.

Il codice

Come già detto, cliccando due volte di seguito su un controllo contenuto nel Form, o sul Form stesso, si apre un'altra finestra, l'**editor di codice**, in cui il programmatore può scrivere le istruzioni necessarie a far compiere al computer determinate operazioni quando si verificano gli eventi. Nella parte superiore dell'editor di codice ci sono due caselle a discesa:







La casella di sinistra fornisce l'elenco degli oggetti disponibili sul Form selezionato, oltre alla sezione **Generale**, che serve per le dichiarazioni, come vedremo oltre; la casella di destra fornisce invece un elenco degli eventi associati al controllo selezionato nella casella di sinistra.

Se si è aperto l'editor del codice cliccando due volte su un controllo (o anche sul form), compaiono l'intestazione e la chiusura della procedura relativa all'evento principale di quel controllo. Ad esempio, se il controllo è il Button di nome btnCalcola, facente parte del form di nome frmImposta, verrà visualizzato, nell'editor:

```
Public Class frmImposta

  Private Sub btnCalcola_Click(ByVal sender As System.Object, ByVal e As
                               System.EventArgs) Handles btnCalcola.Click

  End Sub

End Class
```

Fra **Public Class** e **End Class** sono racchiuse tutte le procedure relative al form e ai controlli disposti su di esso. Il nome del form compare affianco a Public Class.

Ogni procedura è delimitata tra **Private Sub** e **End Sub** (Sub sta per Subroutine, che vuol dire sottoprogramma o procedura).
Di seguito a **Private Sub** figurano:

- il nome dell'oggetto a cui è riferita la procedura e, separato dall'underscore, l'evento a cui la procedura risponde;
- una serie di parametri tra parentesi tonde, due dei quali sono comuni a tutti gli oggetti ed eventi:
  - **sender** identifica l'oggetto su cui si è verificato l'evento;
  - **e** identifica l'evento che si è verificato e porta con sé informazioni ad esso collegate.
- la specifica **Handles**, che definisce l'oggetto e l'evento gestiti dalla procedura.

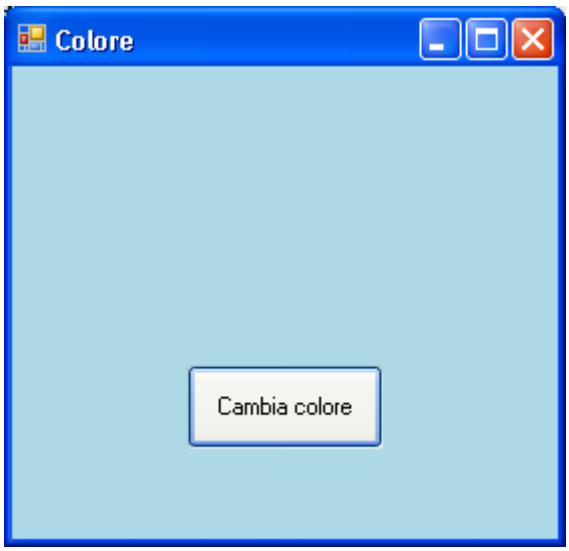
Tra Private Sub e End Sub saranno scritte le istruzioni che gestiscono l'evento.

Nella scrittura del codice, il programmatore può contare su tre importanti supporti:

- la tecnologia **IntelliSense**, che facilita la scrittura di codice fornendo importanti informazioni durante la digitazione. **IntelliSense** è una forma di completamento automatico che riesce a velocizzare lo sviluppo del software, riducendo i dati che il programmatore deve inserire attraverso la tastiera. Riduce inoltre il bisogno di appoggiarsi a documentazione esterna, dato che molte informazioni appaiono in automatico;
- la **formattazione automatica**, cioè il testo del codice viene automaticamente formattato man mano che viene inserito da tastiera, rendendo maiuscole le prime lettere delle **parole-chiave**, regolarizzando gli spazi, usando diversi colori per il testo e per le parole chiave. In particolare:
  - il testo del codice è scritto normalmente in colore nero,
  - le parole-chiave del linguaggio sono indicate con il colore azzurro,
  - i commenti (frasi di commento al codice; sono precedute da un apostrofo e possono collocarsi alla fine di un'istruzione o occupare un'intera riga) sono scritti in verde,
  - le stringhe sono di colore rosso;
- l'**indentazione automatica** del codice (l'incolonnamento delle istruzioni);
- la **chiusura automatica dei costrutti** di selezione e iterazione;
- il **controllo sintattico**, che effettua il controllo sul codice durante l'attività di editing (scrittura e modifica del codice) e segnala gli errori di sintassi, sottolineando con tratto ondeggiato le parole contenenti errori e facendo apparire una tooltip indicativa dell'errore, quando ci si posiziona su una delle parole evidenziate.

Quando nel codice è necessario far riferimento al form di cui fa parte l'oggetto a cui il codice è riferito (quello di cui si sta programmando l'evento), si può usare **Me** al posto del nome del form.

`Es.: Cliccando sul pulsante, cambia il colore del form.`



```
Public Class frmColore
      Private Sub btnCambiaColore_Click(...) Handles btnCambiaColore.Click
        Me.BackColor = Color.LightBlue
    End Sub
End Class
```

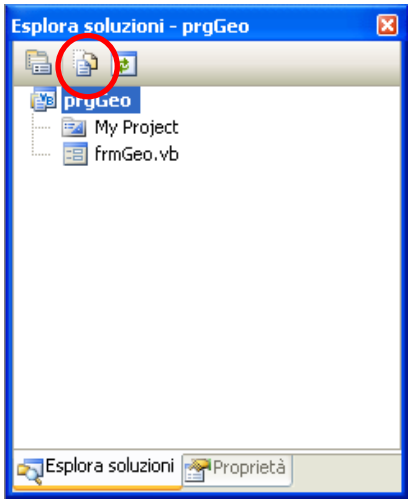
Se una riga di codice è troppo lunga, si può andare a capo, chiudendo la riga precedente con spazio e underscore, nel punto in cui si taglia l'iscrizione.

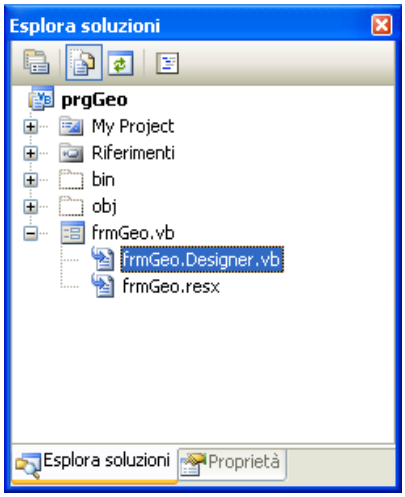
Es.: la lunga intestazione del sottoprogramma all'esempio precedente può essere scritta così

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

Il Form Designer

La gestione dei form avviene per mezzo del Form Designer, un tool per disegnare, modificare e ridimensionare i form, inserire i controlli su di essi, ecc. Il Form Designer genera automaticamente il codice, che può essere comunque visto e modificato dall'utente cliccando sul pulsante **Mostra tutti i file** (**Show all files**) della finestra delle soluzioni:





Nella finestra Esplora soluzioni appaiono tutti i file e le cartelle che fanno parte del progetto corrente. Essi sono normalmente nascosti, perché non gestiti direttamente dall'utente. Questo non vuol dire che il programmatore non possa intervenire su di essi, ma deve farlo con consapevolezza, onde evitare di produrre danni al progetto. I file sono nascosti proprio per evitare modifiche accidentali.
Cliccando sul pulsantino **+** che affianca a sinistra il file .vb di un form, compare il corrispondente file di codice generato dal Form Designer, avente estensione .Designer.vb. Cliccando due volte su questo file, compare una finestra aggiuntiva con il codice generato dal Form Designer; essa si aggiunge alle altre finestre aperte nella parte centrale dell'IDE e può essere chiusa con la consueta **X** in alto a destra.
Un altro dei file che compare ha estensione .resx: è un file di risorse per il form corrispondente. Un file di risorse può contenere stringhe, immagini, suoni, file, icone ed altri tipi di oggetti.

Esecuzione del programma e debugging

Per assicurarsi che l'applicazione sia priva di errori, è importante verificarne la funzionalità in diversi casi. In Visual Basic sono disponibili strumenti che consentono di individuare gli errori e di operare con le opportune modifiche e correzioni. L'attività di individuazione e correzione degli errori viene comunemente indicata in informatica con il termine **debugging** (cioè spulciare, togliere gli errori).
Il programmatore ha innanzitutto la possibilità di eseguire il programma, per verificarne il funzionamento. L'esecuzione avviene, all'interno della finestra di VB, selezionando **Debug** – **Avvia** dalla barra dei menu o cliccando sul pulsante **Avvia debug** ( ).

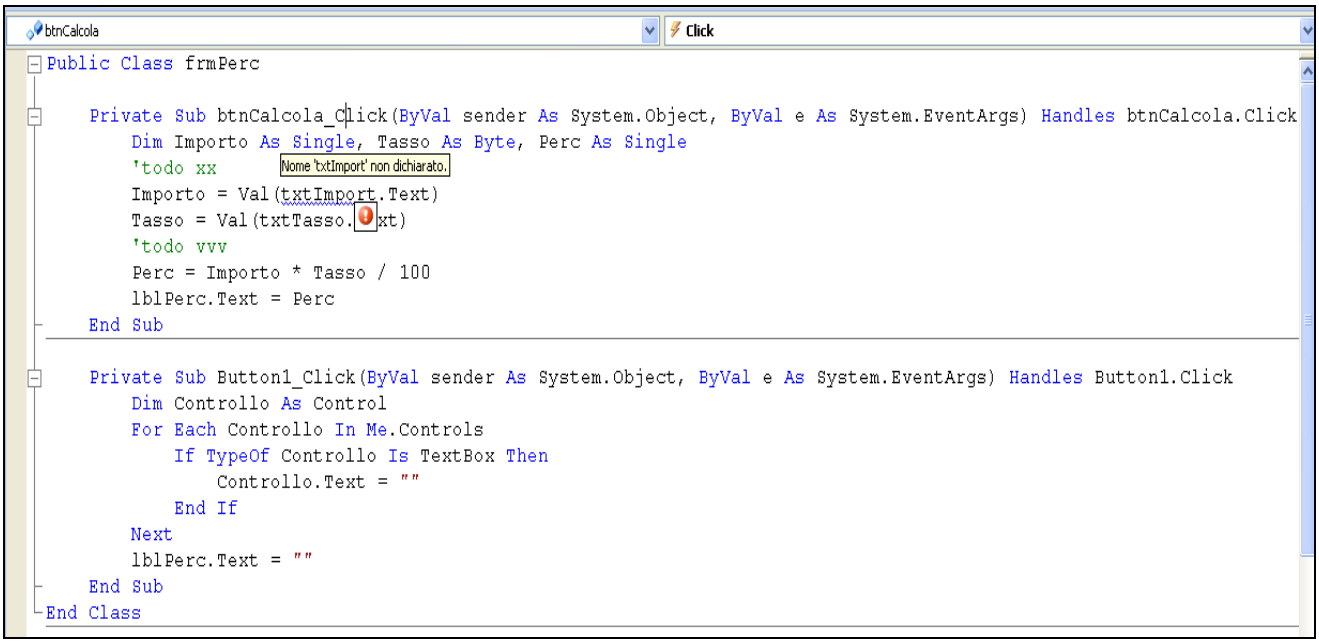
I pulsanti **Interrompi tutto** ( ) e **Termina debug** ( ), o le voci corrispondenti nel menu **Debug**, permettono, rispettivamente, di interrompere temporaneamente l'esecuzione (che può essere ripresa cliccando nuovamente su **Avvia debug**) o di terminarla definitivamente.

Durante l'attività di programmazione si possono commettere tre tipi di errore:

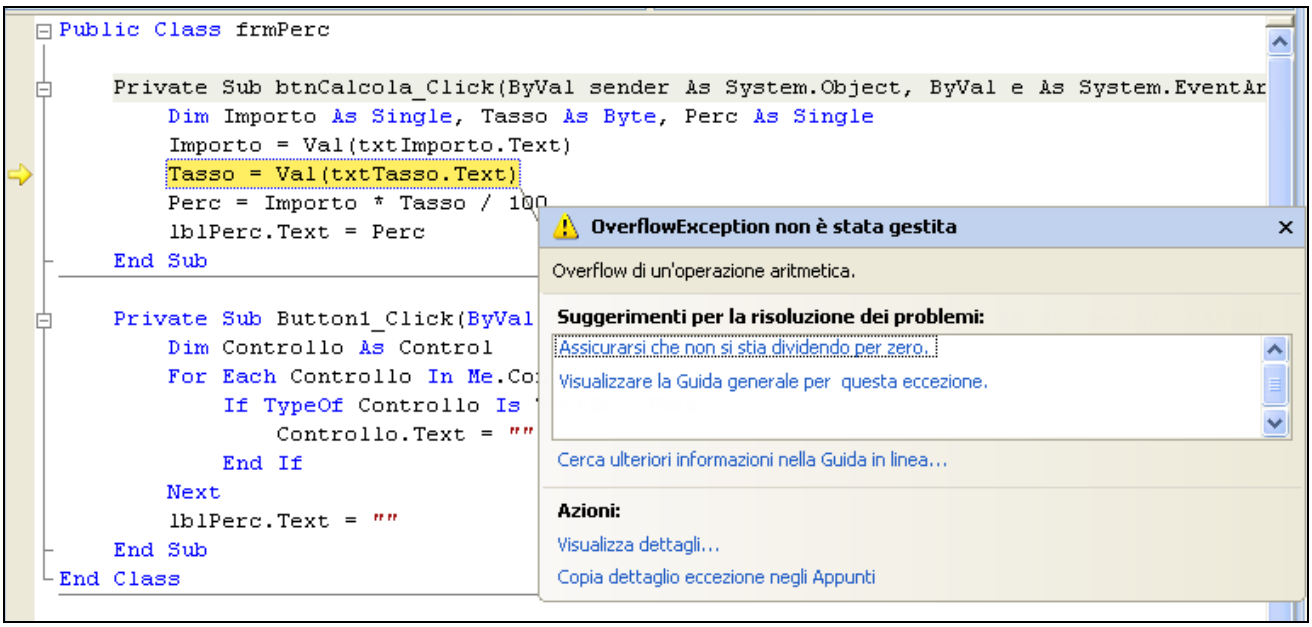
- **errori di compilazione**, causati dall'uso di parole che non appartengono al linguaggio oppure dalla costruzione non corretta di frasi e istruzioni del codice;
- **errori di run-time**, segnalati durante l'esecuzione del programma;
- **errori logici**, che generano risultati diversi da quelli attesi.

Gli **errori di compilazione** sono causati dall'uso di codice Visual Basic in modo non conforme alle regole del linguaggio: possono essere **errori di tipo lessicale**, riguardanti l'uso di parole-chiave, oppure **errori di tipo sintattico**, riguardanti la costruzione delle frasi.

Quando si immettono le istruzioni nell'**Editor di codice**, Visual Basic verifica la sintassi di ogni riga. Se viene individuato un errore, la parte dell'istruzione contenente l'errore viene sottolineata e, sorvolandola con il mouse, compare una tooltip (una sottile striscia gialla) con l'indicazione della possibile causa dell'errore.



Gli **errori di run-time** vengono individuati quando l'applicazione è eseguita (ad esempio a causa di un dato inserito erroneamente dall'utente). Questi errori in VB 2008 prendono il nome di **eccezioni**.
Se si verifica una situazione di errore, l'esecuzione viene interrotta ed è visualizzato un messaggio che indica il tipo di errore e fornisce suggerimenti per risolvere il problema.



Il programmatore ha la possibilità di correggere l'errore e quindi proseguire l'esecuzione dell'applicazione, cliccando sul pulsante **Avvia debug** della barra degli strumenti, oppure può interrompere l'esecuzione con un clic sul pulsante **Termina debug** della barra degli strumenti e riavviare successivamente l'esecuzione.

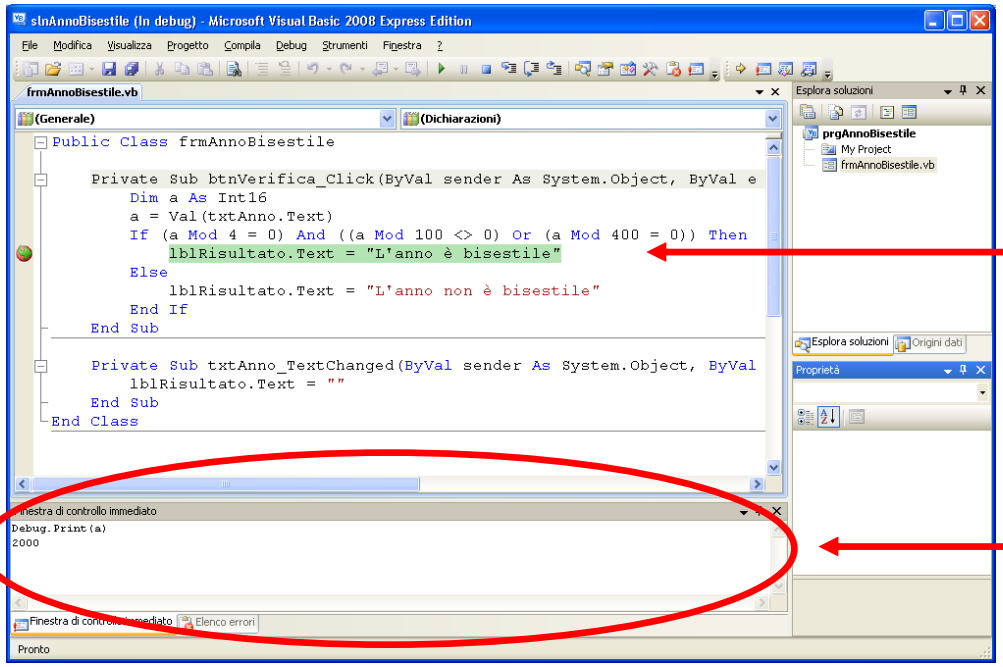
Gli **errori logici** sono i più difficili da individuare; essi si verificano in genere quando l'applicazione in esecuzione produce risultati inattesi. Un esempio di errore logico è una formula scritta correttamente, ma diversa da quella che risolve il problema. Per individuarli, molto comoda è la possibilità offerta dall'opzione **Imposta/Rimuovi punto di interruzione** del menu **Debug** (o tasto F9 o clic nello spazio grigio a sinistra delle istuzioni), che consente di fare in modo che l'esecuzione del programma si fermi quando si raggiunge l'istruzione su cui è stato impostato il punto di interruzione, in modo da poter proseguire l'**esecuzione passo-passo**, selezionando **Esegui istruzione** dal menu **Debug** (o premendo il **tasto F8**) per passare da un'istruzione alla successiva.

```
Public Class frmPerc

    Private Sub btnCalcola_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCalcola.Click
        Dim Importo As Single, Tasso As Byte, Perc As Single
        Importo = Val(txtImporto.Text)
        Tasso = Val(txtTasso.Text)
        Perc = Importo * Tasso / 100
        lblPerc.Text = Perc
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim Controllo As Control
        For Each Controllo In Me.Controls
            If TypeOf Controllo Is TextBox Then
                Controllo.Text = ""
            End If
        Next
        lblPerc.Text = ""
    End Sub
End Class
```

Quando si esegue l'applicazione, l'IDE viene impostata sulla **Modalità di esecuzione**.
In modalità di esecuzione non è possibile apportare modifiche nella **Finestra di progettazione del form**, nella finestra **Proprietà** o in **Esplora soluzioni**. È possibile modificare il codice nell'editor di codice solo in alcuni casi.
In modalità di esecuzione, si può utilizzare la finestra **Controllo immediato**, nella parte inferiore dell'IDE, per valutare variabili ed espressioni, eseguire istruzioni, e così via. Per visualizzare la finestra **Controllo immediato**, selezionare **Espressioni di controllo** dal menu **Debug**, quindi selezionare **Controllo immediato**.
Per utilizzare la finestra Immediata durante l'esecuzione del codice, bloccare l'esecuzione in un punto specifico ed effettuare una delle seguenti operazioni:

- digitare un'istruzione o un metodo, quindi premere INVIO.
- digitare **Debug.Print** seguito d un nome di variabile o da un'espressione.

Es.:

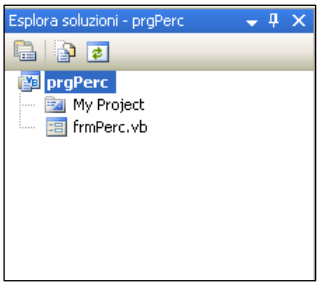


L'esecuzione è sospesa in corrispondenza dell'istruzione evidenziata.

Nella finestra di controllo immediato si visualizza il contenuto della variabile a, usando l'istruzione **Debug.Print(a)**.

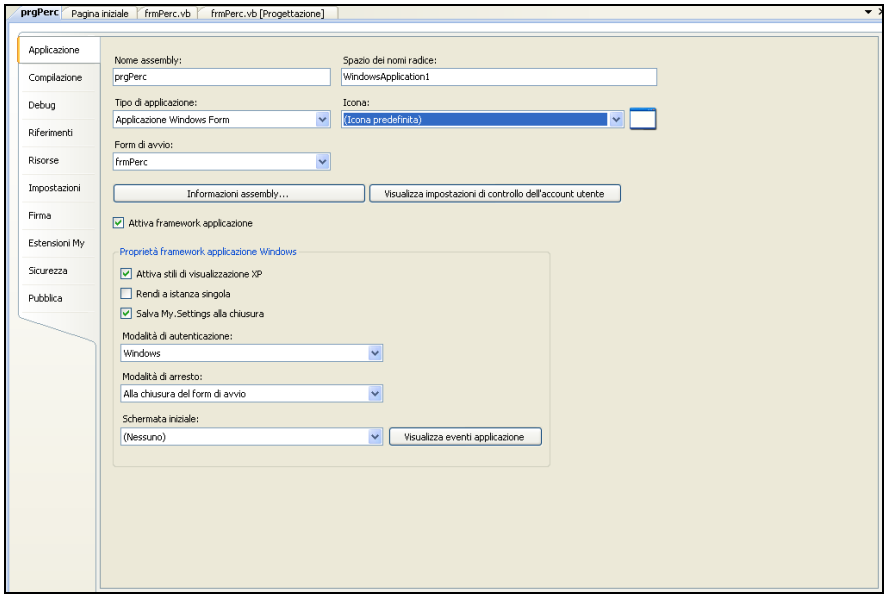
Salvataggio di un progetto

Si è detto che un programma in Visual Basic 2008 è formulato come una Soluzione articolata in uno o più progetti, come mostra la finestra Esplora soluzioni.



Ogni progetto include:

- una cartella, denominata **My Project**, che contiene alcuni documenti con impostazioni relative al progetto. Cliccando due volte su My Project, compare la seguente finestra, dove è possibile modificare tutte le impostazioni del progetto.



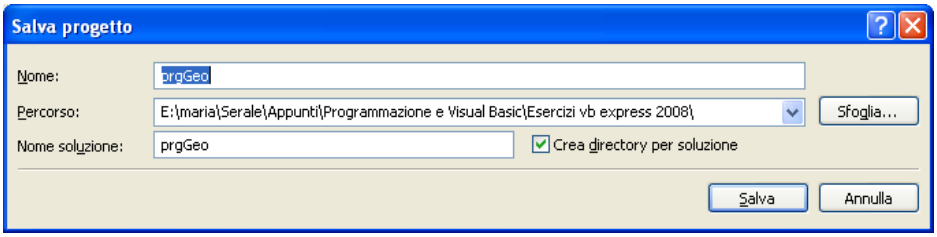
  La stessa finestra compare anche cliccando con il destro sul nome del progetto (nell'esempio, prgPerc) e selezionando Proprietà dal menu a tendina che compare.

- uno o più file con estensione **.vb**, uno per ogni form dell'interfaccia (di cui contiene tutte le proprietà, le informazioni sugli oggetti e il codice) e per ogni altro componente con codice (come i cosiddetti moduli, file contenenti solo codice).

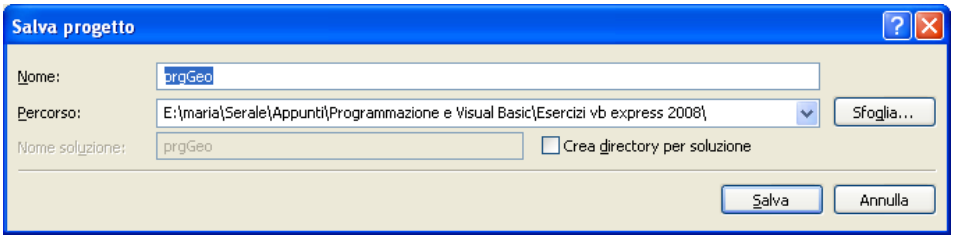
Quando si entra in VB 2008 per creare un nuovo progetto del tipo **Applicazione Windows Form**, viene automaticamente creato un progetto di nome WindowsApplication1, contenente un Form di nome Form1.vb.
È possibile, anzi opportuno, cambiare sia il nome del Form che il nome del progetto impostandone la proprietà **Name** nella **Finestra delle Proprietà**.
Anche in questo caso si può adottare la notazione ungherese, usando, ad esempio, **frm** come prefisso per i form e **prg** per i progetti. Non è possibile invece cambiare il nome della cartella My Project.

Per salvare una soluzione con tutti i suoi progetti, si può procedere nel seguente modo (si consideri, per semplicità, una soluzione contenente un solo progetto):

- Selezionare, dal menu **File**, l'opzione **Salva tutto** oppure cliccare sul pulsante della barra degli strumenti.
- Appare la finestra di dialogo **Salva progetto**.

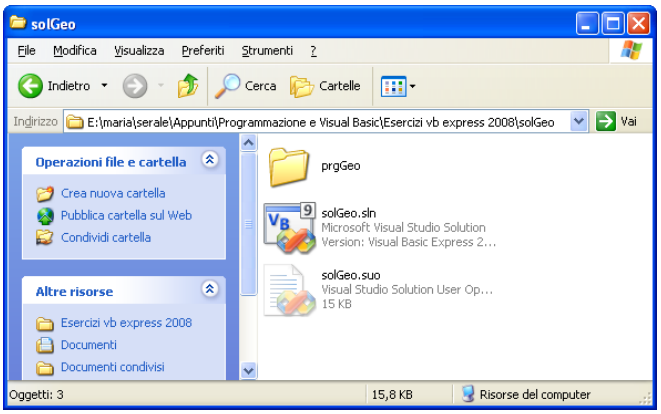


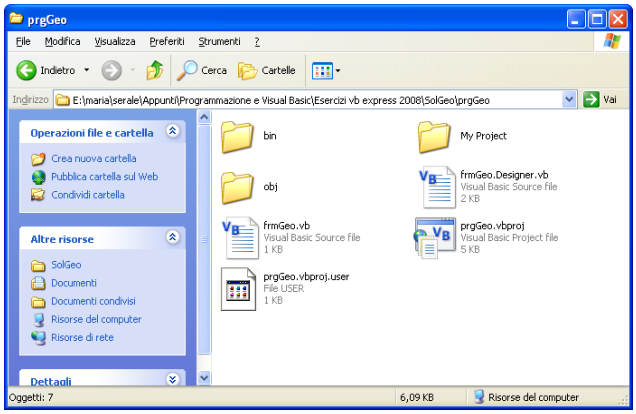
- Nella casella **Nome:** compare il nome del progetto, che sarà attribuito automaticamente al file che lo conterrà; può comunque essere modificato.
- Nella casella **Percorso:** compare il percorso in cui salvare; la prima volta che si salva, appare il percorso predefinito di VB 2008, le volte successive appare l'ultimo percorso in cui si è effettuato il salvataggio. Cliccando il pulsante **Sfoglia...** si può cambiare il percorso.
- Nella casella **Nome soluzione:** compare il nome del progetto, che viene attribuito automaticamente alla soluzione; può ovviamente essere modificato, anzi conviene farlo, per non confondere il progetto con la soluzione.
- Se si spunta la check box **Crea directory per soluzione**, si crea una cartella che contiene i file della soluzione e una cartella con il progetto. Se non si spunta la check box, tutti i file (della soluzione e del progetto) saranno contenuti nella stessa cartella. Il nome della cartella è, nel primo caso, quello dato alla soluzione, nel secondo è quello del progetto (perché in questo caso non c'è possibilità di dare nome alla soluzione).



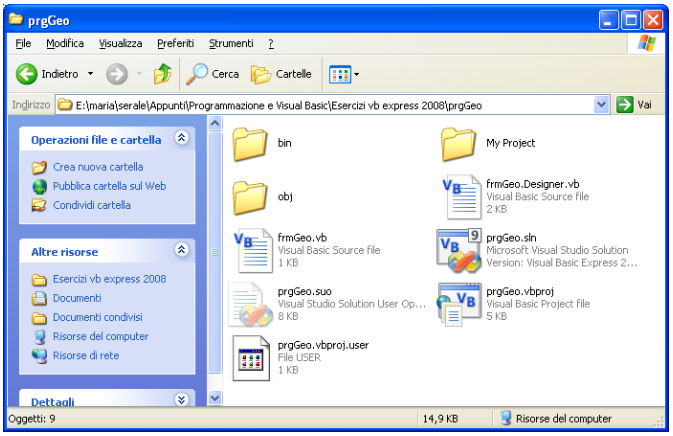
- Si clicca su **Salva**.

Se è stata creata una cartella per la soluzione (SolGeo, nell'esempio), il contenuto della cartella della soluzione e della cartella del progetto (prgGeo, nell'esempio) in essa contenuta sono di seguito riportati.





Se non è stata creata una cartella per la soluzione, c'è solo la cartella del progetto (prgGeo, nell'esempio) e il suo contenuto è di seguito riportato.



I file e le cartelle che figurano in entrambi i casi sono i seguenti, continuando con i nomi dati nell'esempio (solGeo = nome soluzione, prgGeo = nome progetto, frmGeo = nome form):

File

solGeo.sln: è la soluzione.

solGeo.suo: contiene le impostazioni personalizzate della soluzione.

prgGeo.vbproj: è un file di progetto, ogni progetto ha un suo file.

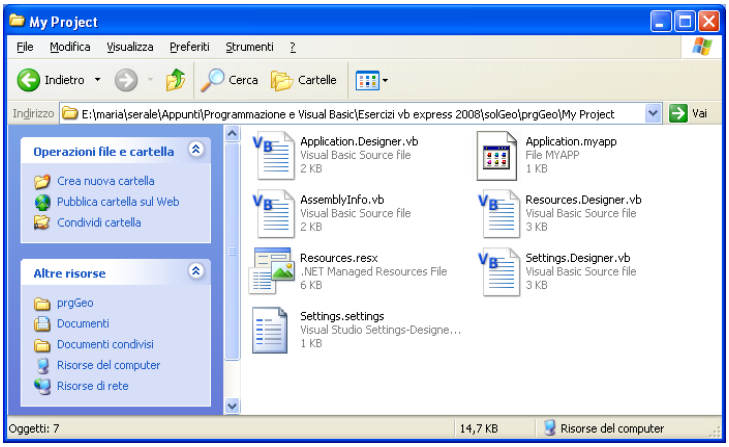
prgGeo.vbproj.user: contiene le impostazioni dell'utente per quel progetto, lo si può cancellare e Visual Basic ne ricrea uno vuoto, appena si cambia qualche impostazione del progetto, VB lo modifica.

frmGeo.vb: è il form.

frmGeo.Designer.vb: contiene codice generato dal FormDesigner di VB.

Cartelle

My Project: come già detto, contiene documenti con impostazioni relative al progetto.



bin: contiene le cartelle Debug e Release. In quest'ultima si trova il codice finale compilato (file .exe) e pronto per la distribuzione.

obj: contiene anch'essa le cartelle Debug e Release, che contengono files intermedi generati dal compilatore, che, in caso di più compilazioni successive, riutilizza i componenti non modificati per accelerare la costruzione del codice oggetto.

Se, dopo il salvataggio, si modifica qualche componente della soluzione, è sufficiente cliccare sul pulsante Salva tutto per registrare le modifiche apportate.
È anche possibile salvare solo un componente (form, progetto) selezionandolo nella finestra **Esplora soluzioni** e cliccando sul pulsante oppure selezionando l'opzione di salvataggio corrispondente dal menu File.

Chiudendo VB, si chiude anche la soluzione.
Per riaprirla, si può:
- aprire VB e selezionare **File - Apri progetto**. Apparirà una finestra in cui è possibile selezionare il file .sln per l'apertura.
- aprire, in risorse del computer, la cartella in cui è stata salvata la soluzione e cliccare due volte sull'icona del file .sln.

Le variabili e i tipi di dati

Anche Visual Basic 2008, come tutti i linguaggi di programmazione, prevede l'uso delle variabili, mediante le quali è possibile memorizzare valori numerici o testuali in aree di memoria centrale a cui il programma può accedere grazie ad un nome assegnato loro in fase di creazione.
Un nome di variabile può contenere solo lettere dell'alfabeto e cifre numeriche; il primo carattere deve essere una lettera. L'unico carattere di altro tipo consentito è il simbolo _ (underscore), usato in genere nei nomi composti da più parole, al posto dello spazio, che non è consentito.
Una variabile è detta **locale** quando è definita all'interno di una procedura; è detta invece **globale** quando è dichiarata al di fuori di tutte le procedure (ad esempio, nell'area generale di un form). La sua creazione avviene quando si fa riferimento ad essa per la prima volta, oppure quando è eseguita l'istruzione **Dim**, che presenta la seguente sintassi:

Dim nome **As** tipo [=valore iniziale]

in cui 'nome' rappresenta il nome da assegnare alla variabile, 'tipo' individua il tipo di dati che essa può contenere. Ad una variabile si può assegnare, facoltativamente, un valore iniziale.
Il tipo può essere uno dei seguenti:

Tipo	Valori	Occupazione in Byte
Boolean	Valori dell'Algebra di Boole.	4
Byte	Numeri interi senza segno da 0 a 255.	1
SByte	Numeri interi con segno da -128 a 127	
Short o Int16	Numeri interi compresi tra -32˙768 e 32˙767.	2
UShort o UInt16	Numeri interi compresi tra 0 e 65˙535.	2
Integer o Int32	Numeri interi compresi fra -2˙147˙483˙648 e 2˙147˙483˙647	4
UInteger o UInt32	Numeri interi compresi fra 0 e 4˙294˙967˙295	4
Long o Int64	Numeri interi compresi fra -923˙372˙036˙854˙775˙808 e 923˙372˙036˙854˙775˙807	8
ULong o UInt64	Numeri interi compresi fra 0 e 18˙446˙744˙073˙709˙551˙615	8
Single	Numeri reali a virgola mobile in semplice precisione. Da -3,402823E38 a -1,401298E-45 per valori Negativi. Da 1,401298E-45 a 3,402823E38 per valori positivi	4
Double	Numeri reali a virgola mobile in doppia precisione. Da -1,79769313486232E308 a -4,94065645841247E-324 per valori negativi. Da 4,94065645841247E-324 a 1,79769313486232E308 per valori positivi.	8
Decimal	Numeri reali a virgola fissa, che sono più precisi nella rappresentazione rispetto ai numeri in virgola mobile. Per la sua precisione (a cui si contrappone la lentezza) è usato per la rappresentazione di valori finanziari, per i quali è richiesto un elevato numero di cifre e non sono tollerati errori di arrotondamento. Sostituisce il tipo currency di VB6.	16
Date	Informazioni temporali, ossia date e ore. Sono rappresentate tutte le date partendo dalle ore 0.00.00 (mezzanotte) del 1° gennaio 0001 alle 23.59.59 del 31 Dicembre 9999. Le date sono rappresentate come numeri: a sinistra della virgola c'è la data, a destra l'ora. Esse seguono il formato anglosassone.	8

String	Sequenze di caratteri alfanumerici UNICODE a lunghezza variabile (ossia numeri, lettere, spazi e caratteri speciali di lunghezza indefinita). Da 0 a circa 2 Miliardi di caratteri.	2 Byte per carattere
Char	Un carattere UNICODE. È usato per evitare di ricorrere al tipo String per un solo carattere.	2
Object	È il tipo di dati in cui vengono trasformate tutte le variabili non dichiarate esplicitamente. Sostituisce il tipo Variant di VB6. Una variabile dichiarata in questo modo potrà memorizzare qualunque tipo di informazioni (tipo di dati universale). In realtà una variabile object non contiene il valore di un dato vero e proprio, ma solo un puntatore al valore. Per questo motivo occupa sempre e solo 4 byte, indipendentemente dal tipo di dati a cui si riferisce. L'accesso alle variabili object risulta però un po' lento, per cui va usato solo quando è necessario.	4

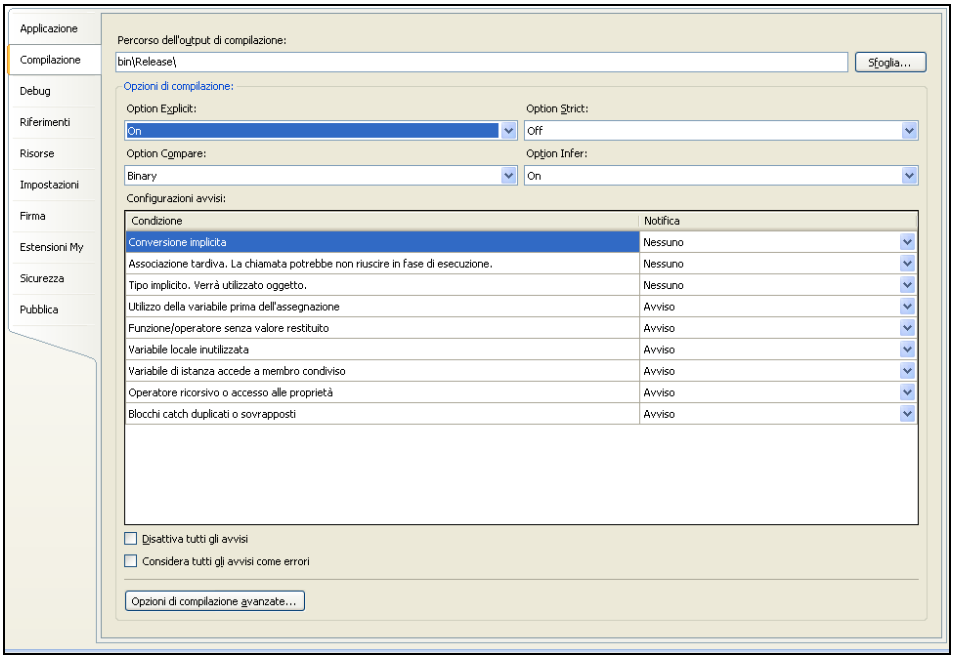
Si può osservare che alcuni tipi sono definiti in versione con segno e senza segno. Al loro nome è anteposta la lettera U se il tipo è senza segno, S se è con segno.
I tipi per i quali si può avere la versione senza segno sono Short, Integer e Long, che quindi compaiono anche nella forma UShort, UInteger e ULong o anche UInt16, UInt32 e UInt64.
Il tipo Byte si può invece anche avere in versione con segno, identificata come SByte.

È possibile dichiarare insieme più variabili aventi lo stesso tipo. Ad esempio:

```
Dim i, j, k As Integer
```

dichiara le variabili i, j, k tutte di tipo Integer.
Non si può usare l'inizializzazione quando c'è una dichiarazione multipla sulla stessa linea.

Anche se non obbligatoria, la dichiarazione delle variabili è fortemente raccomandata. In Visual Basic la si può imporre nella finestra delle proprietà del progetto, alla voce **Compilazione**. L'opzione **Option Explicit**, che può assumere i valori **On** (**dichiarazione delle variabili obbligatoria**) e **Off** (**dichiarazione delle variabili non obbligatoria**). Si osservi che, per default, l'impostazione è On.



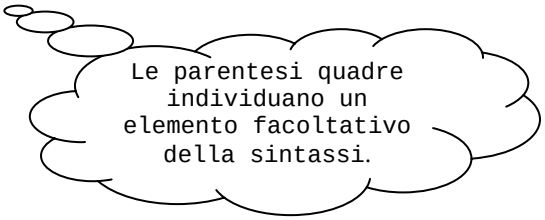
Le variabili non dichiarate sono segnalate, in fase di editing, attraverso una sottolineatura colorata ed una tooltip (che compare quando si passa con il mouse sul suo nome) che riporta un messaggio esplicativo. Ciò consentirà anche di scoprire errori di digitazione commessi nella scrittura dei nomi delle variabili.

Le costanti

Se nell'applicazione è contenuto un valore che non viene mai modificato, è possibile definire una costante implicita (vale a dire un nome simbolico) e utilizzarla al posto di un valore esplicito. È più facile ricordare un nome che un valore. È possibile definire la costante una sola volta e utilizzarla in numerose posizioni del codice.
Per definire una costante si usa la parola chiave **Const**, con la seguente sintassi:

Const nome [**As** tipo] =valore

```
Es.:
        Const Maxi As Short = 459
    oppure
        Const Maxi = 459
```



Istruzioni di assegnazione

Le istruzioni di assegnazione consentono di assegnare valori alle variabili. La sintassi è

Variabile = valore

```
Es.:    N = 5
        Nome = ”Giulia”
      La variabile N è di tipo numerico, la variabile Nome è di tipo stringa.
      “Giulia” è invece una costante stringa esplicita.
```

Si usa un’istruzione di assegnazione anche per assegnare ad una variabile il valore digitato dall’utente in una TextBox (input di dati). Se la variabile è numerica, occorre effettuare una conversione da stringa a numero mediante la funzione **Val**, perché il tipo di valore contenuto in una TextBox è considerato di tipo stringa.

```
Es.:  N = Val(txtNumero.Text)
      Nome = txtNome.Text
      Anche in questo caso, la variabile N è di tipo numerico, la variabile Nome è
      di tipo stringa.
```

Anche per assegnare ad una proprietà di un oggetto un valore (ad esempio ad una Label, per un’operazione di output), si usa l’operazione di assegnazione.

```
Es.:  lblRisultato.Text = N
      lblResponso.Text = “Il numero è pari”
      La prima istruzione assegna alla Label lblRisultato il valore contenuto
      nella variabile N. Se N è numerica, non serve conversione.
      La seconda istruzione visualizza nella Label lblResponso il messaggio “Il
      numero è pari”.
```

In generale, nelle istruzioni che devono far riferimento al valore di una proprietà di un controllo si scrive il nome del controllo seguito da un punto e dal nome della proprietà (txtNumero.Text, lblRisultato.caption, ecc.). Lo stesso dicasi per i metodi dei controlli (txtNumero.Clear svuot la TextBox txtNumero)

Operatori aritmetici

In un programma è possibile eseguire tutte le operazioni aritmetiche attraverso i seguenti operatori:

+	addizione
-	sottrazione
*****	moltiplicazione
^	elevamento a potenza
/	divisione reale (produce eventuali cifre decimali)
****	divisione intera (si ferma alla parte intera del quoziente)
mod	resto della divisione tra due interi

Es.:

```
q = 9 / 2 assegna alla variabile q il valore 4.5
q = 9 \ 2 assegna alla variabile q il valore 4
r = 9 mod 2 assegna alla variabile r il valore 1
```

Nelle espressioni matematiche si possono utilizzare solo parentesi tonde, anche nidificate a più livelli.

L'operatore di elevamento a potenza può essere usato anche per l'estrazione di radice, in quanto vale la regola matematica:

$$\sqrt[n]{x} = x^{1/n}$$

Ad esempio, la radice quadrata di un numero si può calcolare con la seguente istruzione:

```
r = X^(1/2)
```

oppure

```
r = X^0.5
```

Analogamente, per la radice cubica si può usare l'istruzione:

```
r = X^(1/3)
```

Solo per la radice quadrata è disponibile anche la funzione **Math.Sqrt**:

```
r = Math.Sqrt (X)
```

Sono anche disponibili gli **operatori di assegnazione con calcolo**, che abbreviano le istruzioni quando ad una variabile si deve assegnare il valore di se stessa modificato mediante una delle operazioni aritmetiche:

Operatore	Esempio
+=	X+=10 equivale a X=X+10
-=	X-=10 equivale a X=X-10
*=	X*=10 equivale a X=X*10
/=	X/=10 equivale a X=X/10
\=	X\=10 equivale a X=X\10

Operatori sulle stringhe

L'operazione più frequente su dati di tipo stringa è la concatenazione, vale a dire la creazione di una nuova stringa accodando due o più stringhe.
Il VB dispone di due operatori di concatenazione: **+** e **&**.
Per entrambi si può utilizzare anche l'assegnazione con calcolo (nelle forme += e &=).

Esempi:

```
Nome = "Mario"
Cognome = "Rossi"
NomeCompleto = Nome & " " & Cognome
```

Oppure, accodando Cognome a Nome:

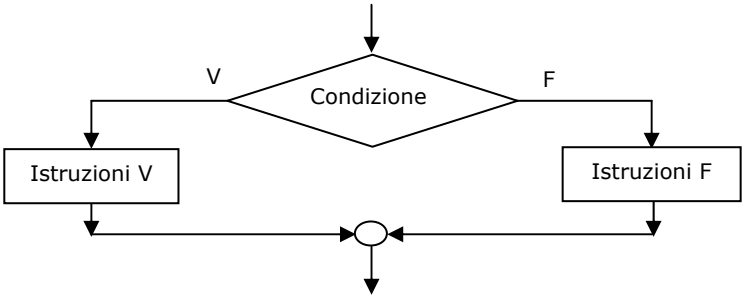
```
Nome &= " " & Cognome
```

Le strutture della programmazione

La struttura di selezione: If … Then … Else

Il flusso delle istruzioni all'interno della procedura di risoluzione di un problema non è rigidamente definito, ma può variare in base al verificarsi o meno di determinate condizioni. Il programma deve pertanto essere in grado di prendere decisioni.
A tal fine si può utilizzare la **struttura di selezione**, caratterizzata dalla valutazione di una condizione booleana, che può essere vera (true) o falsa (false). Se la condizione è vera si segue un percorso, se è falsa se ne segue un altro. Al termine di entrambi i percorsi, le strade si uniscono.

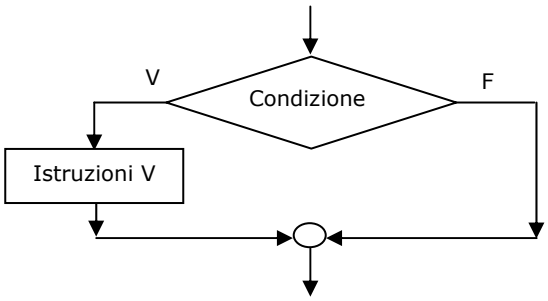
Nel diagramma di flusso la struttura di selezione è così rappresentata:



Visual Basic codifica la struttura di selezione con il costrutto **If … Then … Else**, caratterizzato dalla seguente sintassi:

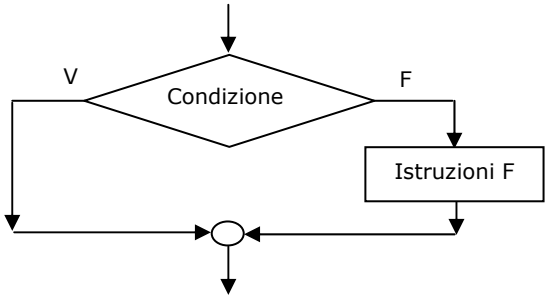
If Condizione **Then**
    Istruzioni da eseguire se la condizione è vera (Istruzioni V)
Else
    Istruzioni da eseguire se la condizione è falsa (Istruzioni F)
End If

Si osservi che le istruzioni dopo **Then** e le istruzioni dopo **Else** rientrano verso destra: è una tecnica di scrittura del codice che prende il nome di **Indentazione** e ha il solo obiettivo di migliorarne la leggibilità.
Uno dei due rami di una selezione può anche essere vuoto. Se è vuoto il ramo corrispondente alla condizione falsa, si può omettere la parola **Else**, per cui il costrutto diventa:



If Condizione **Then**
    Istruzioni da eseguire se la condizione è vera
End If

Se invece il ramo vuoto corrisponde alla condizione vera, il costrutto diventa:



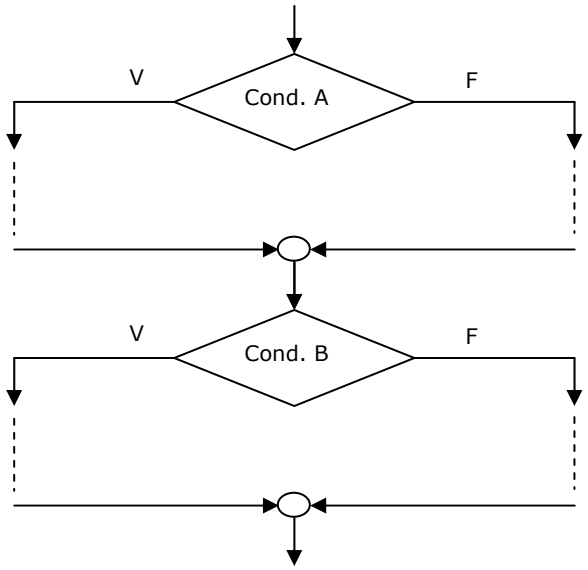
If Condizione **Then**
Else
    Istruzioni da eseguire se la condizione è falsa
End If

La condizione di una selezione può essere:

- un confronto fra una coppia di dati attraverso gli operatori relazionali o di confronto: >, <, =, >=, <=, <>.
- la combinazione di due o più confronti attraverso gli **operatori booleani o logici**: **And**, **Or**, **Not**. L'operatore **And** restituisce il valore logico Vero solo se le condizioni a cui è applicato sono contemporaneamente verificate. L'operatore **Or** restituisce il valore logico Vero se almeno una delle condizioni specificate è vera. L'operatore **Not** restituisce il valore Vero se la condizione a cui è applicato non è verificata, mentre restituisce Falso in caso contrario.

Le strutture di selezione possono anche essere combinate:
- in cascata
- nidificate (o annidate).

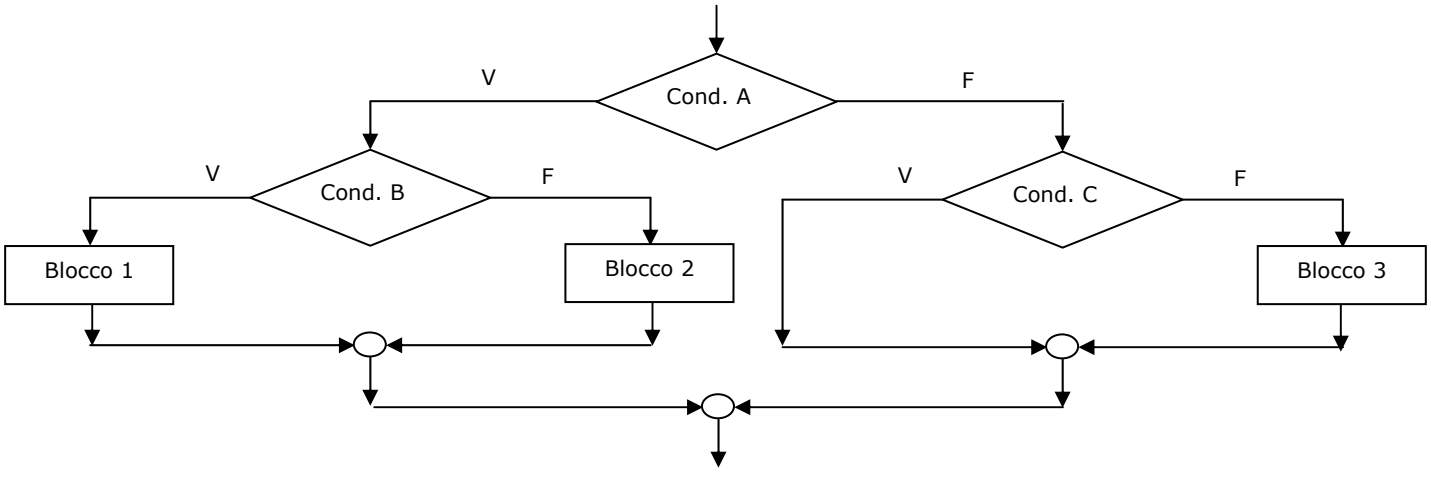
Le strutture di selezione in cascata sono semplici sequenze di selezioni:




```
If Cond.A Then
  …..
Else
  …
End If

If Cond.B Then
  …..
Else
  …
End If
```

Come si osserva dal codice nel riquadro a destra della figura, l'accorgimento da osservare nella codifica è solo quello di aprire il successivo costrutto If solo dopo la chiusura del precedente.

Le strutture di selezione nidificate, comunemente molto utilizzate, si basano sulla considerazione che sia sul ramo "vero" che sul ramo "falso" è possibile inserire un'altra struttura di selezione, che a sua volta ne può contenere altre, e così via. L'accorgimento è di non far mai incrociare le linee delle strutture e di chiudere prima le strutture interne, poi quelle più esterne, e così via.
Un esempio di strutture di selezione nidificate è il seguente:



Il codice della struttura presentata è il seguente:

```
If Cond. A Then

    If Cond. B Then
        Blocco 1
    Else
        Blocco 2
    End If

Else

    If Cond. C Then
    Else
        Blocco 3
    End If

End If
```

ElseIf e Select Case

Una struttura If con varie nidificazioni può assumere dimensioni notevoli. In questi casi, la leggibilità del programma rischia di essere gravemente intaccata. Per evitare che questo accada, è possibile ricorrere alla struttura **If … Then … ElseIf … Else … End If**, che è così articolata:

```
If condizione1  Then
      istruzioni da eseguire se condizione1 è vera
ElseIf              condizione2                    Then
      istruzioni da eseguire se condizione2 è vera
ElseIf              condizione3                    Then
      istruzioni da eseguire se condizione3 è vera
…
Else
      istruzioni da eseguire se tutte le condizioni sono false
End If
```

Questo costrutto può essere utilizzato solo se sul ramo falso della prima condizione c'è subito un'altra condizione, sul cui ramo falso c'è subito un'altra condizione, e così via. In tal caso ogni If si "attacca" all'Else della condizione precedente e tutte le condizioni nidificate condividono lo stesso End If.

Anche la struttura **Select Case** consente di implementare selezioni con molte nidificazioni.
Può essere utilizzata ogniqualvolta si desideri variare il flusso del programma in base ai valori che una variabile può assumere. Ogni valore deve essere preceduto dalla parola chiave **Case**; segue il gruppo di istruzioni da eseguire quando la variabile assume quel valore. Inoltre è possibile utilizzare la clausola **Case Else**, che va posta alla fine della struttura, per definire un gruppo di istruzioni che deve essere eseguito solo se tutti i confronti hanno ottenuto un esito negativo. Il codice risulta in questo caso più leggibile.

```
Select Case variabile
    Case valore1
        blocco istruzioni 1
    Case valore2
        blocco istruzioni 2
    . . .

    Case valoreN
        blocco istruzioni N
    Case Else
        istruzioni da eseguire se tutti i confronti falliscono
End Select
```

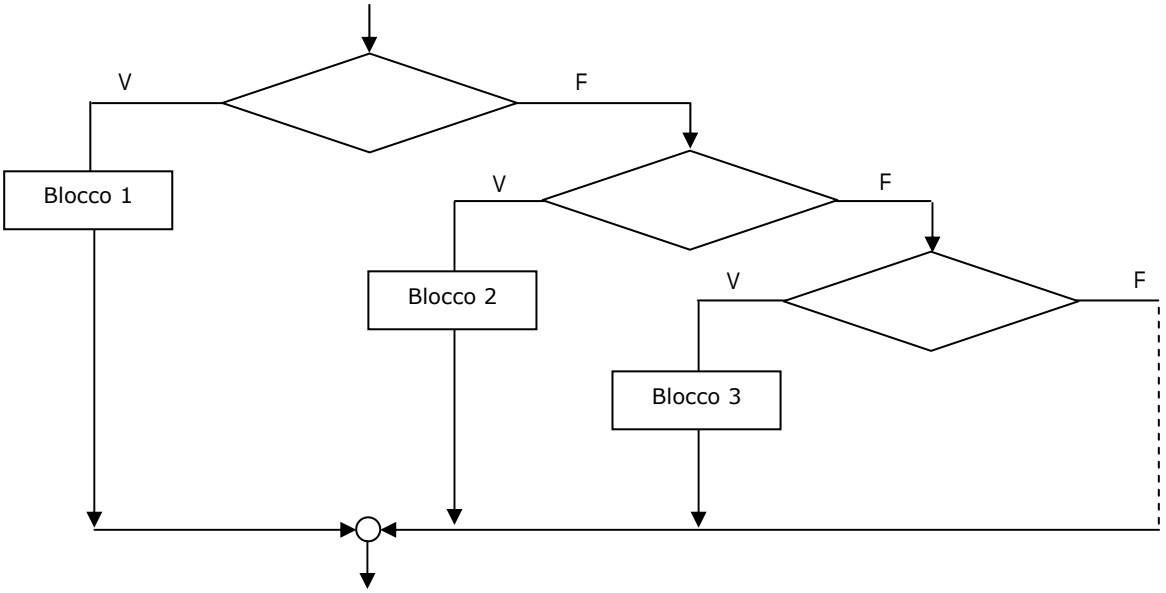
È anche possibile:

- elencare una serie di valori, separati da virgola, con cui confrontare la variabile in Select Case,
- indicare intervalli di valori, specificati dai due estremi inframmezzati dalla parola chiave **To**,
- utilizzare gli operatori di confronto, con la parola chiave **Is**, seguita da un'espressione di confronto,

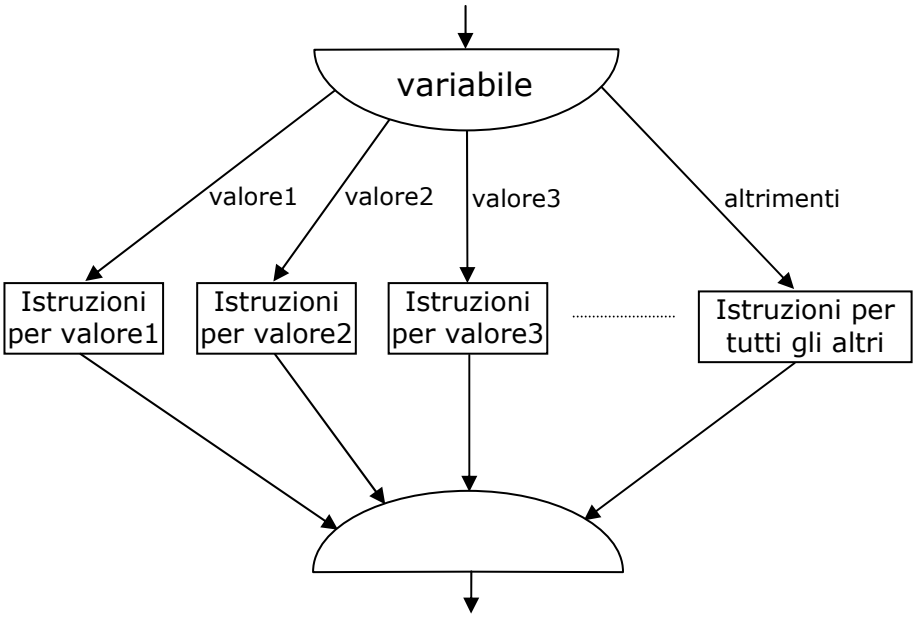
come negli esempi indicati di seguito.

```
Select Case V
        Case 1, 2, 3
                ...
        Case 4 To 10
                ...
        Case Is > 10
                ...
        Case Else
                ...
End Select
```

Entrambe queste strutture non sono strutture standard della programmazione, ma sono state introdotte dai linguaggi per semplificare le istruzioni. Non esiste pertanto un formalismo standard nel diagramma di flusso, che ognuno può "ideare" a suo piacimento.
Per la struttura If...ElseIf...Else...End If si può ad esempio usare la seguente rappresentazione:



Mentre per Select...Case si può usare la seguente:



Le strutture iterative

Nella formulazione dell'algoritmo di risoluzione dei problemi molto spesso si presenta la necessità di ripetere più volte una stessa sequenza di operazioni. Si impone allora il ricorso alle **strutture di iterazione** (o **strutture iterative**), dette comunemente anche **cicli**.

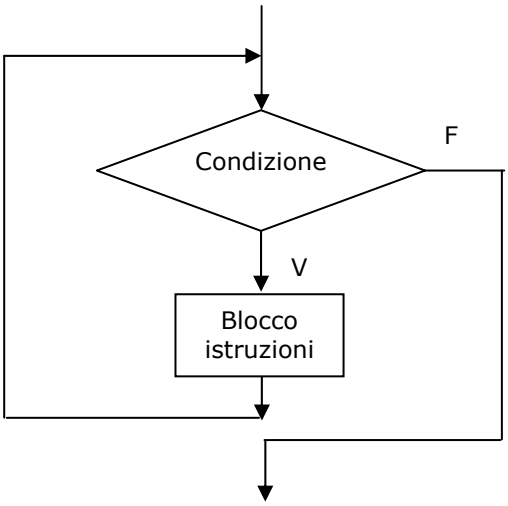
Le istruzioni che il ciclo ripete prendono il nome di **corpo del ciclo.**

Tre sono le strutture iterative fondamentali:

- **Ciclo precondizionale**

  La condizione è posta in testa al ciclo, che permette la ripetizione di un blocco di istruzioni per tutto il tempo in cui la condizione risulta vera. Il ciclo termina quando la condizione diventa falsa.

  Si osservi che, se la condizione è subito falsa, le istruzioni del ciclo non vengono mai eseguite.



La sintassi è una delle seguenti:

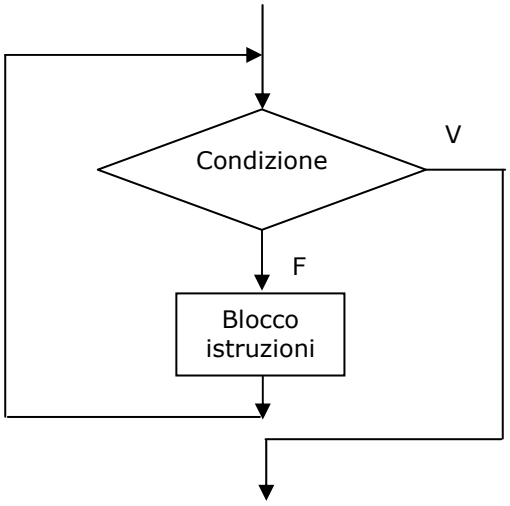
```
Do While condizione
        Blocco istruzioni
Loop
```

oppure

```
While condizione
        Blocco istruzioni
End While
```

Le istruzioni del ciclo sono ripetute per un numero di volte non stabilito a priori, ma 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 incorre in un errore detto **ciclo infinito**. Gli effetti di un ciclo senza fine sono evidenti: il programma si blocca.

Una variante (costrutto non standard) di questa struttura prevede che il ciclo venga ripetuto se la condizione è falsa e termini se la condizione è vera.



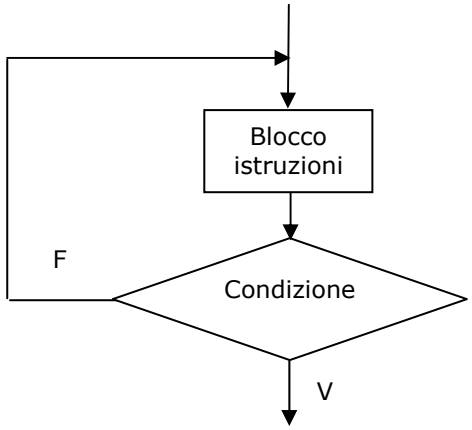
La sintassi è:

```
Do Until condizione
        Blocco istruzioni
Loop
```

Esegue le istruzioni specificate fino a che non viene soddisfatta la condizione, ma se la condizione è valida all'inizio, non viene eseguita nessuna istruzione del blocco.

- **Ciclo postcondizionale**

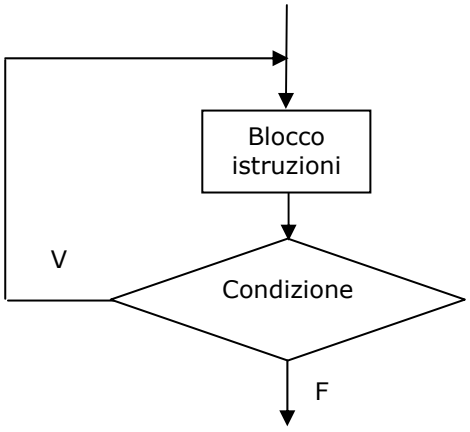
La condizione che regola il ciclo è posta in coda al ciclo stesso, permettendo la ripetizione di un blocco di istruzioni per tutto il tempo in cui la condizione risulta falsa. Il ciclo termina quindi quando la condizione diventa vera.



La sintassi è la seguente:

```
Do
        Blocco istruzioni
Loop Until condizione
```

Con questo tipo di struttura le istruzioni del ciclo vengono eseguite almeno una volta, a differenza di quanto accade con il ciclo precondizionale.
Una variante (costrutto non standard) di questa struttura prevede che il ciclo venga ripetuto se la condizione è vera e termini se la condizione è falsa.



La sintassi è la seguente:

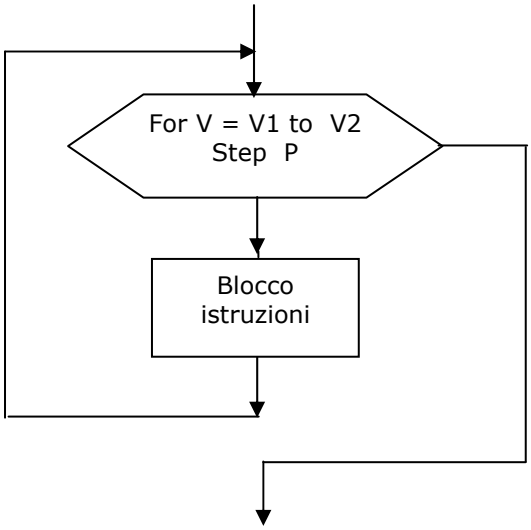
```
Do
        Blocco istruzioni
Loop While condizione
```

Esegue le istruzioni specificate fintanto che una condizione rimane valida, ma tutte le istruzioni vengono eseguite almeno una volta.

- **Ciclo enumerativo**

La struttura enumerativa permette la ripetizione di una porzione di codice per un numero finito di volte, precisamente per tutti i valori che una variabile I, detta **variabile (o indice) del ciclo**, assume da un valore iniziale V1 a un valore finale V2, con un certo incremento P, denominato **passo** (facoltativo; se non specificato si assume 1).
Se V1 è superiore a V2, il passo deve essere negativo, affinché il ciclo possa essere eseguito almeno una volta.

Questa struttura iterativa non è standard, per cui la sua rappresentazione nel diagramma di flusso non è univocamente riconosciuta. Si può, ad esempio, utilizzare la seguente:



La sintassi nel linguaggio Basic è la seguente:

```
For I = V1 To V2 [Step P]
        Blocco istruzioni
Next
```

Quando I raggiunge V2, il ciclo termina e il controllo passa all'istruzione successiva a Next.

Es.: Sommare tutti i numeri pari tra 8 e 40.

```
For I = 8 To 40 Step 2
     Somma = Somma + I
Next
```

La struttura enumerativa non è sempre applicabile, in quanto presuppone che si sappia in anticipo il numero di volte in cui il ciclo deve essere ripetuto, cosa che in realtà non sempre avviene; in generale, infatti, è il verificarsi o meno di una condizione che stabilisce quando le ripetizioni devono avere termine.
I cicli precondizionali e postcondizionali consentono, al contrario, di realizzare qualunque iterazione.

Per tutti i cicli è possibile realizzare strutture nidificate (ciclo dentro ciclo, anche di tipo diverso).

Istruzione Exit

Consente di uscire prematuramente da un ciclo e di trasferire immediatamente il controllo all'istruzione successiva alla definizione del ciclo. Si usa scrivendo il nome del costrutto iterativo (Do, While, For) dopo la parola Exit.

Exit Do
Esce immediatamente dal ciclo Do nel quale è inserita. L'esecuzione continua con l'istruzione successiva all'istruzione Loop.

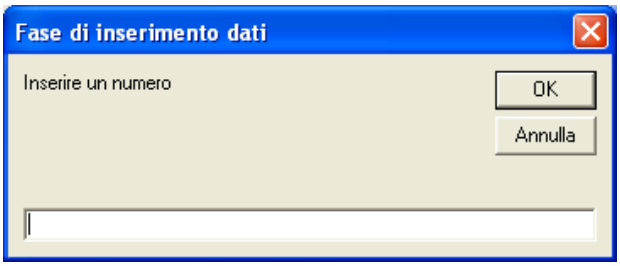
Exit For
Esce immediatamente dal ciclo For nel quale è inserita. L'esecuzione continua con l'istruzione successiva all'istruzione Next.

Exit While
Consente di uscire immediatamente dal ciclo While nel quale è inserita. L'esecuzione continua con l'istruzione successiva all'istruzione End While.

Se utilizzata all'interno di cicli nidificati, Exit esce dal ciclo più interno e trasferisce il controllo al livello superiore più prossimo di nidificazione.

Finestre di dialogo predefinite di VB 2008: InputBox

La funzione InputBox viene utilizzata per la richiesta di dati. Consente infatti di visualizzare una finestra di dialogo a scelta obbligatoria in cui viene richiesto di immettere i dati.



La sintassi della funzione InputBox è la seguente:

Variabile=InputBox (Prompt, Titolo)

o, se la variabile è numerica,

Variabile=Val(InputBox (Prompt, Titolo))

dove:

- *Prompt* è un'espressione stringa che costituisce il messaggio visualizzato nella finestra di dialogo. la lunghezza massima è pari a circa 1024 caratteri e dipende dalla larghezza dei caratteri utilizzati. Se include più righe, è possibile separare le varie righe tramite un carattere di ritorno a capo Chr (13).
- *Titolo* è un'espressione stringa visualizzata nella barra del titolo della finestra di dialogo.

Ad esempio, per la finestra in figura, il codice è il seguente:

```
N = Val (InputBox ("Inserire un numero", "Fase di inserimento dati")
```

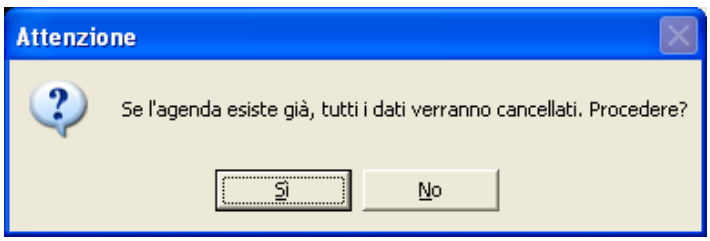
L'utente deve digitare il dato nella casella di testo e cliccare su **OK** o premere **INVIO**, perché il dato sia acquisito. Il pulsante Annulla deve invece essere programmato da codice, ad esempio nel seguente modo:

```
Dim S As String, N As Byte
        S = InputBox("Inserire un numero :", " Fase di inserimento dati")
        If IsNothing(S) = 0 Then
            MsgBox("Input annullato")
        Else
            N = Val(S)
            MsgBox("Input accettato")
        End If
```

La funzione IsNothing restituisce il valore True se alla variabile S non è stato assegnato un valore, cosa che avviene se si clicca sul pulsante Annulla anziché su OK, qualunque cosa sia stata scritta nella casella di testo dell'InputBox.
Se l'input avviene all'interno di un ciclo, si può aggiungere, al codice di gestione del pulsante Annulla, l'uscita forzata dal ciclo.

Finestre di dialogo predefinite di VB 2008: MsgBox

Visualizza un messaggio in una finestra di dialogo e, dopo che l'utente ha cliccato su un pulsante, restituisce un valore che indica il pulsante scelto.



Può essere usato come procedura o come funzione.

Per inviare semplicemente un messaggio, la sintassi è:

MsgBox (Prompt, Icona + Pulsanti, Titolo)

Per inviare un messaggio ed acquisire la risposta su quale pulsante l'utente ha cliccato, il risultato deve essere assegnato ad una variabile e i parametri devono essere racchiusi tra parentesi tonde:

R = MsgBox (Prompt, Icona + Pulsanti, Titolo)

dove:

- *Prompt* e *Titolo* hanno il significato già visto per InputBox.
- *Icona + Pulsanti* è un'espressione numerica corrispondente alla somma dei valori che specificano il numero e il tipo di pulsanti da visualizzare, lo stile di icona da utilizzare e il pulsante da utilizzare come impostazione predefinita.

Le possibili impostazioni dell'argomento *Icona + Pulsanti* sono:

	Costante	**Valore**	**Descrizione**
Icona	MsgBoxStyle.Critical	16	Visualizza l'icona di messaggio critico.
	MsgBoxStyle.Question	32	Visualizza l'icona di richiesta di avviso.
	MsgBoxStyle.Exclamation	48	Visualizza l'icona di messaggio di avviso.
	MsgBoxStyle.Information	64	Visualizza l'icona di messaggio informativo.
Numero e tipo di pulsanti	MsgBoxStyle.OKOnly	0	Visualizza solo il pulsante OK.
	MsgBoxStyle.OKCancel	1	Visualizza i pulsanti OK e Annulla.
	MsgBoxStyle.AbortRetryIgnore	2	Visualizza i pulsanti Interrompi, Riprova e Ignora.
	MsgBoxStyle.YesNoCancel	3	Visualizza i pulsanti Sì, No e Annulla.
	MsgBoxStyle.YesNo	4	Visualizza i pulsanti Sì e No.
	MsgBoxStyle.RetryCancel	5	Visualizza i pulsanti Riprova e Annulla.
Pulsante predefinito	MsgBoxStyle.DefaultButton1	0	Il primo pulsante è il predefinito.
	MsgBoxStyle.DefaultButton2	256	Il secondo pulsante è il predefinito.
	MsgBoxStyle.DefaultButton3	512	Il terzo pulsante è il predefinito.

Il primo gruppo di valori, 16, 32, 48, 64, individua l'icona visualizzata nella finestra di dialogo.
Il secondo gruppo, da 0 a 5, individua il numero e il tipo dei pulsanti visualizzati nella finestra di dialogo.
Il terzo gruppo 0, 256, 512, indica qual è il pulsante predefinito.
Quando si sommano i numeri per la creazione di un valore finale dell'argomento *Icona + Pulsanti*, si deve utilizzare soltanto un numero per ciascun gruppo.

Ad esempio, per il messaggio nella pagina precedente l'istruzione è:

```
R = MsgBox ("Se l'agenda esiste già, tutti i dati saranno cancellati. Procedere?",
                                          vbYesNo + vbInformation, "Attenzione")
```

La funzione MsgBox restituisce i seguenti valori:

Costante	Valore	Pulsante scelto
MsgBoxResult.OK	1	OK
MsgBoxResult.Cancel	2	Annulla
MsgBoxResult.Abort	3	Interrompi
MsgBoxResult.Retry	4	Riprova
MsgBoxResult.Ignore	5	Ignora
MsgBoxResult.Yes	6	Sì
MsgBoxResult.No	7	No

Se nella finestra di dialogo è disponibile un pulsante Annulla, premendo ESC si potrà ottenere lo stesso risultato del pulsante Annulla.

Ad esempio, il messaggio nella pagina precedente restituirà il valore 6 se l'utente clicca sul pulsante Sì o il valore 7 se l'utente clicca sul pulsante No.

MsgBox era già disponibile, con la sintassi vista, in VB6. In VB2008 è disponibile anche la nuova classe **MessageBox**.
La sintassi è:

MessageBox.Show (Prompt, Titolo, Pulsanti, Icona)

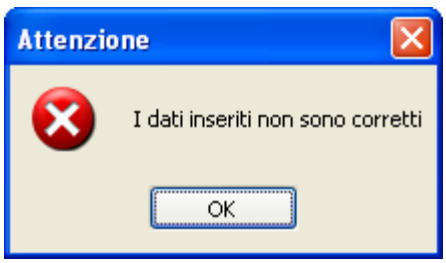
Quindi c'è una diversa indicazione dei parametri. Cambiano anche gli identificatori di pulsanti e icone:

	Costante	Descrizione
Numero e tipo di pulsanti	MessageBoxButton.OK	Visualizza solo il pulsante OK.
	MessageBoxButton.OKCancel	Visualizza i pulsanti OK e Annulla.
	MessageBoxButton.AbortRetryIgnore	Visualizza i pulsanti Interrompi, Riprova e Ignora.
	MessageBoxButton.YesNoCancel	Visualizza i pulsanti Sì, No e Annulla.
	MessageBoxButton.YesNo	Visualizza i pulsanti Sì e No.
	MessageBoxButton.RetryCancel	Visualizza i pulsanti Riprova e Annulla.
Icona	MessageBoxIcon.Error	Visualizza l'icona di messaggio critico.
	MessageBoxIcon.Question	Visualizza l'icona di richiesta di avviso.
	MessageBoxIcon.Exclamation	Visualizza l'icona di messaggio di avviso.
	MessageBoxIcon.Information	Visualizza l'icona di messaggio informativo.
	MessageBoxIcon.None	Nessuna icona

Es.:

```
MessageBox.Show("I dati inseriti non sono corretti", "Attenzione", _
                                MessageBoxButtons.OK, MessageBoxIcon.Error)
```

Visualizza il messaggio:



I principali controlli di VB 2008

Gruppo Controlli comuni

Label

È uno dei controlli più semplici, perché la maggior parte delle sue proprietà è dedicata alla configurazione del suo aspetto e perché permette di visualizzare un testo fisso, modificabile solo da codice.

Tra le proprietà più utili:

- **Text**: è il testo contenuto nella label.
- **TextAlign**: definisce l'allineamento del testo nella label.
- **Image**: carica un'immagine nella label. La proprietà **ImageAlign** permette di definire il tipo di allineamento dell'immagine.
- **Autosize**: fa sì che la label si dimensioni automaticamente, adattandosi al contenuto.

TextBox

È uno dei controlli maggiormente utilizzati. Permette l'inserimento di un testo qualsiasi.

Tra le proprietà più utili:

- **Text**: è il testo contenuto nella casella.
- **Maxlength**: limita il numero di caratteri che possono essere inseriti.
- **PasswordChar**: se si vuole inserire testo nascosto, assegnare a questa proprietà il carattere che si vuole far apparire durante la digitazione (ad esempio, un asterisco).
- **TextAlign**: definisce l'allineamento del testo nella casella.
- **Multiline**: true se si vuole che nella casella si possano inserire più righe di testo, false (default) in caso contrario.
- **ScrollBars**: fa apparire o meno le barre di scorrimento, nel caso in cui il testo superi le dimensioni della casella.

Tra gli eventi più utili:

- **TextChanged**: si verifica quando viene modificato il contenuto della casella di testo.

Il metodo **Clear**, infine, cancella il contenuto della casella di testo.

```
Es.  txtNumero.Clear()
```

Button

È anche questo un controllo molto utilizzato, perché è su un pulsante che in genere l'utente clicca per eseguire operazioni, tant'è che il Button trova posto praticamente in tutte le interfacce.

Alcune importanti proprietà sono:

- **Text**: è il testo che compare sul pulsante. Anteponendo il carattere **&** ad un carattere del testo, tale carattere compare sottolineato e il tasto corrispondente diventa tasto di accesso, per cui, premere ALT e il tasto equivale a cliccare sul pulsante.
- **TextAlign**: definisce l'allineamento del testo nella casella (MouseDownBackColor).
- **BackgroundImage**: attribuisce un'immagine di sfondo al pulsante. Il testo digitato nella proprietà Text appare in sovraimpressione all'immagine. La proprietà **BackGroundImageLayout** permette di definire il tipo di allineamento dell'immagine e se si deve estendere (Stratch) all'intero pulsante.
- **FlatStyle**: può assumere uno di 4 stati, che sono *Standard* (di default), *Popoup*, *System* e *Flat*. Lo stile Flat utilizza la proprietà **FlatAppearance**, che fornisce alcune proprietà per personalizzare il controllo, quali colore (*BorderColor*) e dimensione (*BorderSize*) del bordo, colore del pulsante quando il mouse si sposta sul controllo (*MouseOverBackColor*), colore del pulsante quando lo si tiene premuto con il mouse (*MouseDownBackColor*).
- **Image**: visualizza un'immagine sul pulsante (non di sfondo). La proprietà **ImageAlign** permette di definire il tipo di allineamento dell'immagine e può essere utile per evitare

che il testo si sovrapponga all'immagine (basta, ad esempio, allineare l'immagine in alto e il testo in basso, o viceversa).

Per comodità dell'utente, i due tasti **INVIO** ed **ESC** della tastiera possono essere associati all'evento click di altrettanti pulsanti del form. Nelle proprietà del form, basta scegliere i nomi dei pulsanti tra quelli disponibili rispettivamente nelle proprietà **AcceptButton** e **CancelButton**.

NumericUpDown

È una speciale TextBox che accetta soltanto un singolo valore numerico che può essere aumentato o diminuito facendo clic sui pulsanti con la freccia rivolta verso l'alto o verso il basso del controllo.

Alcune importanti proprietà sono:

- **Value**: è il valore contenuto nella casella.
- **TextAlign**: definisce l'allineamento del testo nella casella.
- **Minimum** e **Maximum**: definiscono il valore minimo e il valore massimo consentiti nella casella.
- **Increment**: é il valore da aggiungere o sottrarre alla proprietà Value quando l'utente fa clic sui pulsanti con la freccia rivolta verso l'alto o verso il basso.
- **ReadOnly**: l'utente può anche inserire un valore nella casella, a meno che questa proprietà non sia impostata su *True*.
- **DecimalPlaces**: per specificare il numero di cifre visualizzate dopo il punto decimale, assegnare a questa proprietà il numero di posizioni decimali da visualizzare.
- **ThousandsSeparator**: per visualizzare un separatore delle migliaia nei numeri decimali, impostare questa proprietà a *True*.
- **Hexadecimal**: per visualizzare valori esadecimali nella casella, impostare questa proprietà a *True*.

RadioButton

Il controllo RadioButton (**pulsante di opzione**) consente di gestire un gruppo di opzioni tra le quali una sola può essere selezionata. Ad ogni opzione è associato un controllo RadioButton.
Per raggrupparli, occorre inserirli all'interno di un contenitore, ad esempio un GroupBox o un Panel. Tutti i controlli RadiopButton di uno stesso contenitore si escludono a vicenda, cioè quando se ne seleziona uno, tutti gli altri controlli del gruppo vengono deselezionati.
Tutti i RadioButton inseriti direttamente in un form fanno parte dello stesso gruppo.
È possibile selezionare un RadioButton in uno dei modi seguenti:

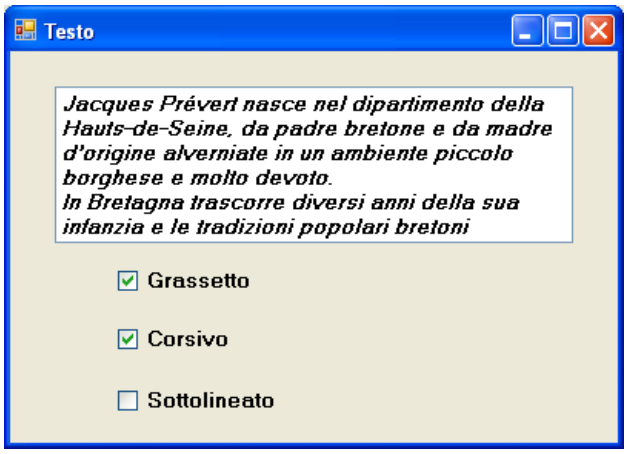
- Fare clic sul pulsante, in fase di esecuzione.
- Impostare a *True* la proprietà **Checked** del pulsante nella finestra delle proprietà o nel codice.

Impostando la proprietà **Checked** a *False*, il pulsante viene invece deselezionato.

La proprietà **Text** di un pulsante di opzione permette di visualizzare un testo accanto al pulsante. La proprietà **TextAlign** permette di definire l'allineamento del testo nella casella.
La proprietà **CheckAlign** permette di definire varie posizioni del pulsante rispetto al testo.
La proprietà **RightToLeft** permette di disporre il testo a destra o a sinistra del pulsante.

La proprietà **Appearance** modifica l'aspetto del controllo per farlo sembrare un interruttore. I valori possibili sono Normal e Button. Quando si clicca sul pulsante, esso cambia colore.
Sono anche disponibili le proprietà **Image** e **ImageAlign**.

CheckBox



Le CheckBox (**caselle di controllo**) indicano se una determinata condizione è attivata o disattivata e consentono di fornire all'utente opzioni di tipo vero/falso o sì/no.
Dato che questi controlli funzionano in modo indipendente l'uno dall'altro, un utente può selezionare più caselle contemporaneamente.
È possibile selezionare una CheckBox in uno dei modi seguenti:

- Fare clic sul controllo, in fase di esecuzione.
- Impostare a *True* la proprietà **Checked** del controllo nella finestra delle proprietà o nel codice.

Impostando la proprietà **Checked** a False, il pulsante viene invece deselezionato.

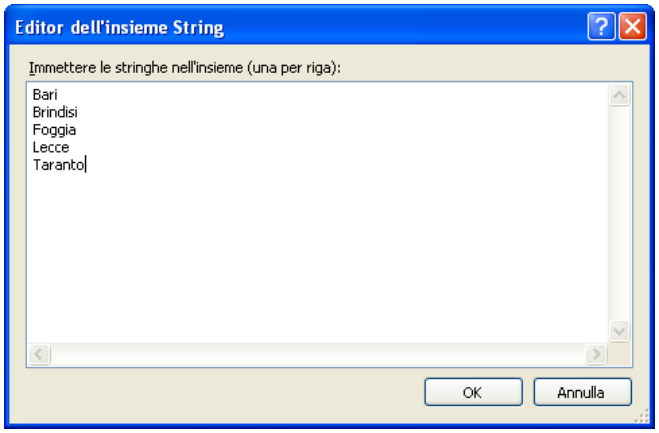
È anche possibile utilizzare una CheckBox a tre stati: selezionato, non selezionato, indeterminato.

Si definisce una CheckBox a tre stati ponendo a True la proprietà **ThreeState**. Solo in tal modo è possibile, cliccando sul controllo in fase di esecuzione, impostare uno di tre stati, anziché due.
Per leggere/impostare lo stato di una CheckBox a tre stati, si deve utilizzare la proprietà **CheckState** (i valori sono Checked, UnChecked, Indeterminate) anziché la proprietà Checked.
La proprietà **Text** permette di visualizzare un testo accanto al pulsante. La proprietà **TextAlign** permette di definire l'allineamento del testo nella casella.
La proprietà **CheckAlign** permette di definire varie posizioni del pulsante rispetto al testo.
La proprietà **RightToLeft** permette di disporre il testo a destra o a sinistra del pulsante.
Sono anche disponibili le proprietà **Image** e **ImageAlign**.

Anche per le CheckBox si possono definire le proprietà **Caption**, **Alignment**, **ForeColor**, **Backcolor** e **Font**.

ListBox

Il controllo ListBox consente di visualizzare un elenco di elementi. Se il numero di elementi dell'elenco è maggiore del numero di elementi visualizzabili a causa delle dimensioni limitate del controllo, viene automaticamente aggiunta una barra di scorrimento.
È possibile, in fase di progettazione, inserire elementi in una ListBox nel seguente modo:



- nella finestra delle proprietà, cliccare sulla proprietà **Items**, che individua l'insieme dei dati presenti nella lista;
- cliccare sul pulsante , che appare a destra;
- nella finestra che appare, digitare il primo elemento da inserire nella lista e premere INVIO per passare alla riga successiva;
- proseguire digitando gli elementi e premendo INVIO;
- Cliccare su OK per terminare e chiudere la finestra.

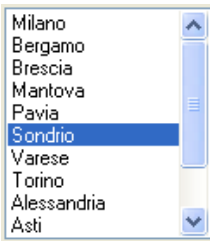
Per aggiungere o eliminare elementi in un controllo ListBox da codice, sono disponibili i metodi **Add**, **Insert**, **Remove**, **Clear** di Items.

Il metodo **Add** aggiunge un elemento ad un controllo ListBox.
```
Es.     lstElenco.Items.Add (S) Aggiunge la stringa S in coda alla lista lstElenco
```

Il metodo **Insert** inserisce un elemento tra quelli già presenti in un controllo ListBox.
```
Es.     lstElenco.Items.Insert (N, S)   Inserisce la stringa S prima dell'elemento
                                         in posizione N nella lista lstElenco
```

Il metodo **Remove** rimuove un elemento da un controllo ListBox.
```
Es.:    lstElenco.Items.Remove S        Rimuove da lstElenco la stringa S.
```

Il metodo **Clear** cancella il contenuto di un controllo ListBox.
```
Es.:    lstElenco.Items.Clear
```



È possibile anche rilevare eventuali selezioni effettuate dall'utente.
La proprietà **SelectedIndex** di un controllo ListBox contiene il numero della posizione dell'elemento selezionato. Gli elementi sono numerati a partire da 0.
Se non è selezionato un elemento, il valore di SelectedIndex è -1.
La proprietà **SelectedItem** di un controllo ListBox contiene l'elemento selezionato.
Usando queste proprietà come parametri di Insert e Remove è possibile Inserire un nuovo elemento prima della posizione selezionata e rimuovere l'elemento nella posizione selezionata.

```
Es.:    lstElenco.Items.Insert(lstElenco.SelectedIndex, S)
Es.:    lstElenco.Items.Remove(lstElenco.SelectedItem)
```

Per consentire l'accesso ai singoli elementi presenti nella lista, si usano invece le proprietà **Item** e **Count** di **Items**.

La proprietà **Count** restituisce il numero di elementi contenuti nella lista; non è disponibile in fase di progettazione.

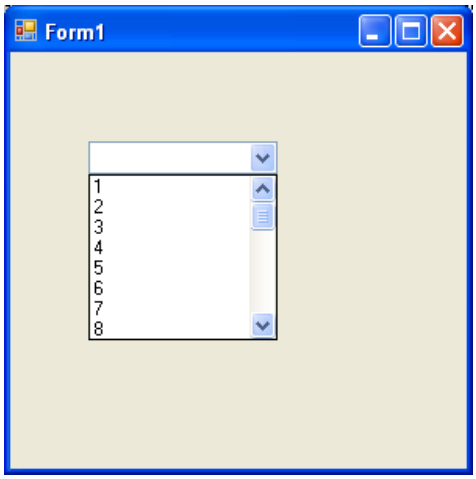
```
Es.: n = lstElenco.Items.Count      assegna a n il numero di elementi nella ListBox
```

La proprietà **Item** restituisce o imposta gli elementi contenuti nell'elenco. Questa proprietà viene usata per accedere ad ogni singolo elemento.

```
Es.: lstElenco.Items.Item(K) = S    pone nella posizione K della lista la stringa S
     S = lstElenco.Items.Item(K)    assegna alla variabile stringa S l'elemento
                                     della lista in posizione K.
```

ComboBox

Il controllo ComboBox (**casella combinata**) unisce le caratteristiche dei due controlli TextBox e ListBox. Valgono quindi tutte le proprietà e i metodi degli altri due controlli, compresa la proprietà Text, che contiene il dato presente nella casella di testo della ComboBox.
Per caricare la tendina della ComboBox si può ricorrere al codice, ad esempio in corrispondenza dell'**evento Activated del form** che si verifica quando il form diventa attivo, in modo che la tendina sia carica quando l'interfaccia appare all'utente.

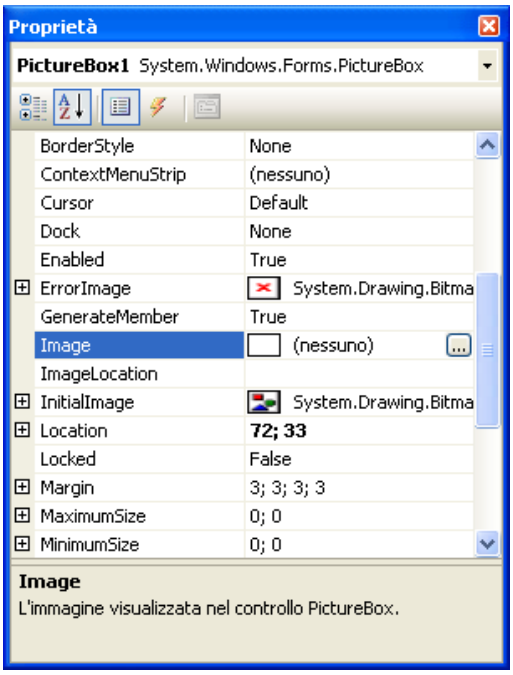



```
Es.: per caricare la tendina con i numeri interi da 1
     a 31, il codice potrebbe essere:
```

```
Private Sub Form1_Activated(...) Handles Me.Activated
        For I = 1 To 31
            cmbNumeri.Items.Add(I)
        Next
End Sub
```

PictureBox

Permette di visualizzare un'immagine, il cui formato può essere BMP, GIF (anche animata), JPG, WMF, EMF, PNG, ICO, CUR.
L'immagine viene assegnata alla proprietà **Image**. Questo può avvenire sia in fase di progettazione, nella finestra delle proprietà, che da codice.
Nella finestra delle proprietà basta selezionare la proprietà Image, cliccare sul pulsantino con tre puntini e selezionare la risorsa (in questo caso l'immagine) da caricare.
La finestra che compare è la stessa che viene visualizzata tutte le volte che deve essere caricata un'immagine:



Per impostare invece da codice la proprietà Image, si scrive
NomePictureBox.Image = System.Drawing.Bitmap.FromFile(PathNameFile)
dove

NomePictureBox è il nome del controllo PictureBox
PathNameFile è il percorso del file che contiene l'immagine.

Esempio:

```
pctImmagine.Image = System.Drawing.Bitmap.FromFile("C:\Immagini\Gattino.jpg")
```

carica l'immagine "Gattino.jpg" nella PictureBox di nome pctImmagine.

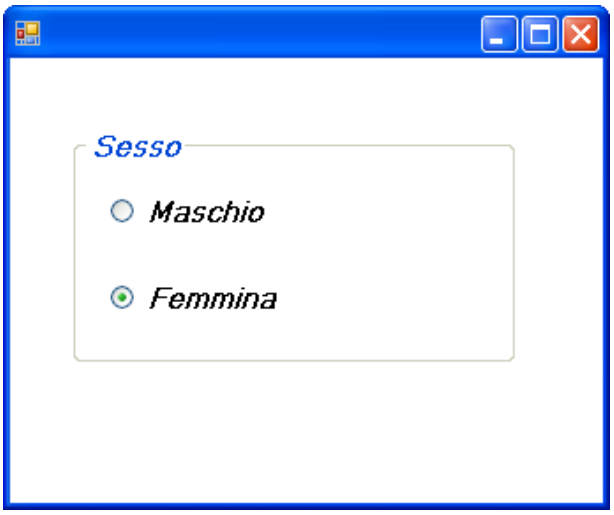
Per eliminare l'immagine da codice, è sufficiente porre la proprietà Image uguale a *Nothing*. Nella finestra delle proprietà basta invece cancellare il valore assegnato alla proprietà Image.

La proprietà **SizeMode** permette di impostare la posizione dell'immagine nella visualizzazione: *Normal* è l'impostazione predefinita; *StretchImage* adatta le dimensioni dell'immagine perché possa essere interamente visualizzata all'interno del controllo; *AutoSize* ridimensiona la PictureBox sulla base della grandezza dell'immagine; *CenterImage* visualizza l'immagine centrata nella PictureBox.

Gruppo Contenitori

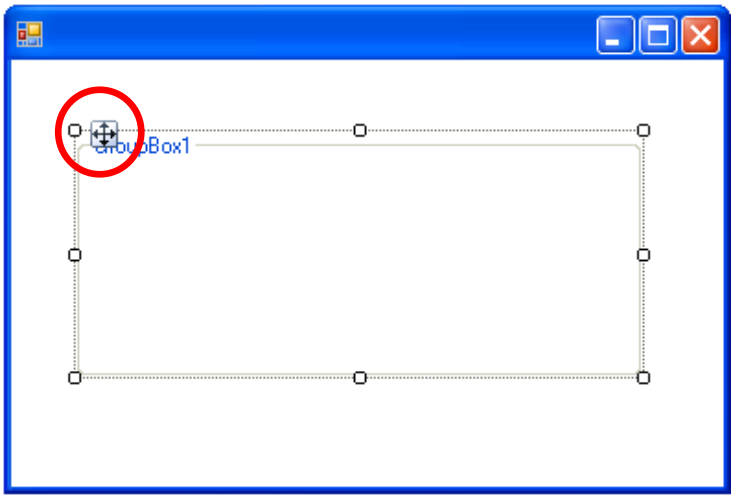
GroupBox

È un contenitore di raggruppamento, generalmente utilizzato per raggruppare più pulsanti di opzione mutuamente esclusivi, ma può contenere qualsiasi altro controllo.
Gli oggetti inseriti al suo interno verranno logicamente considerati insieme, come facenti parte di un unico gruppo: ad esempio, nascondendo un controllo GroupBox, verranno automaticamente nascosti anche tutti i controlli in esso contenuti.
Per inserire oggetti in un controllo GroupBox è sufficiente selezionarli e trascinarli all'interno del controllo stesso; analogamente, se si vuole portare un oggetto all'esterno del GroupBox, basta selezionarlo e spostarlo al di fuori di esso.



Se si sposta un controllo GroupBox verranno ovviamente spostati anche tutti gli oggetti al suo interno. Per lo spostamento, si deve utilizzare il piccolo pulsante in alto a sinistra, cliccando su di esso e trascinando il mouse.
Le proprietà più utilizzate sono:

- **Caption** consente di attribuire al GroupBox un titolo.
- **BackColor** imposta il colore dell'interno del GroupBox.
- **ForeColor** imposta il colore del titolo del GroupBox.
- **Font** imposta lo stile, il tipo e la dimensione dei caratteri del titolo.
- **BackgroundImage**: si usa per impostare un'immagine di sfondo. La proprietà **BackGroundImageLayout** permette di definire il tipo di allineamento dell'immagine e se si deve estendere (*Stretch*) all'intero GroupBox.



Panel

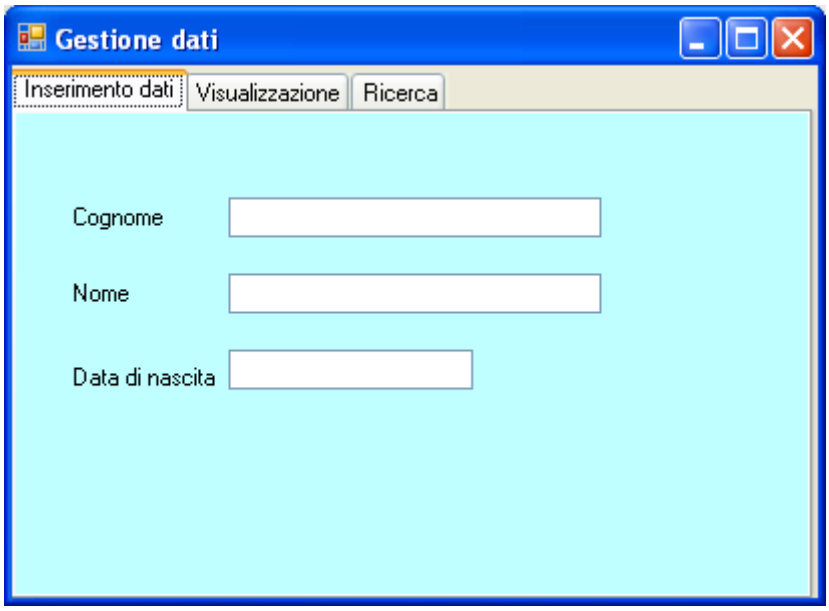
Il controllo Panel è molto semplice da utilizzare. È un altro controllo contenitore, cioè consente di definire un'area al cui interno è possibile inserire altri oggetti logicamente considerati come facenti parte di un unico gruppo.
Le considerazioni fatte relativamente al GroupBox si applicano anche al Panel. La differenza tra i due è che il GroupBox definisce un'area delimitata e visibile in fase di esecuzione, mentre i bordi del controllo Panel sono invisibili a runtime (in esecuzione).

TabControl

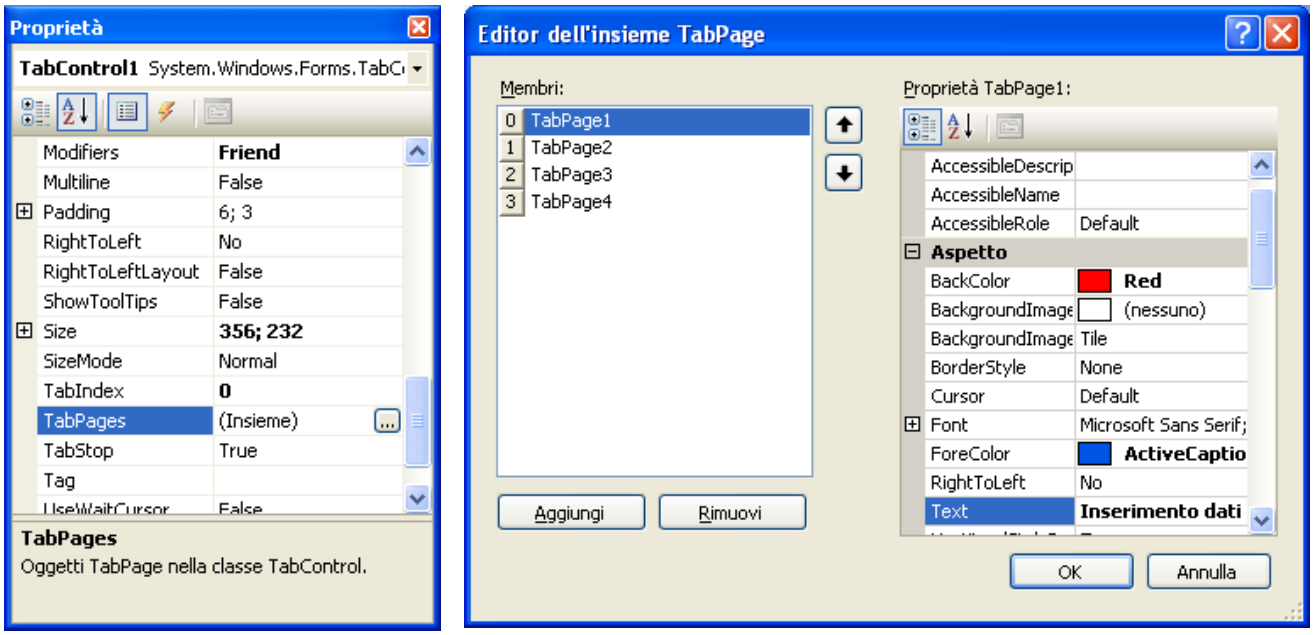
Si presenta all'utente come un insieme di contenitori a schede. Ogni scheda può contenere un diverso insieme di controlli. Per inserire uno o più componenti di una scheda è sufficiente selezionare la scheda desiderata ed inserire i controlli come si fa normalmente. Si noterà che, passando da una scheda all'altra, verranno visualizzati solo i controlli appartenenti a ciascuna scheda.

Il vantaggio di questo contenitore è il notevole guadagno in termini di spazio: è come avere più Form nello spazio occupato da un solo Form.

È consigliabile estendere il TabControl in modo che occupi l'intero Form.



Quando l'oggetto viene posizionato sul Form, ha solo due schede. Per aggiungere una nuova scheda, selezionare nella finestra delle proprietà la proprietà **TabPages**. Apparirà una finestra di dialogo dove è possibile personalizzare il controllo.

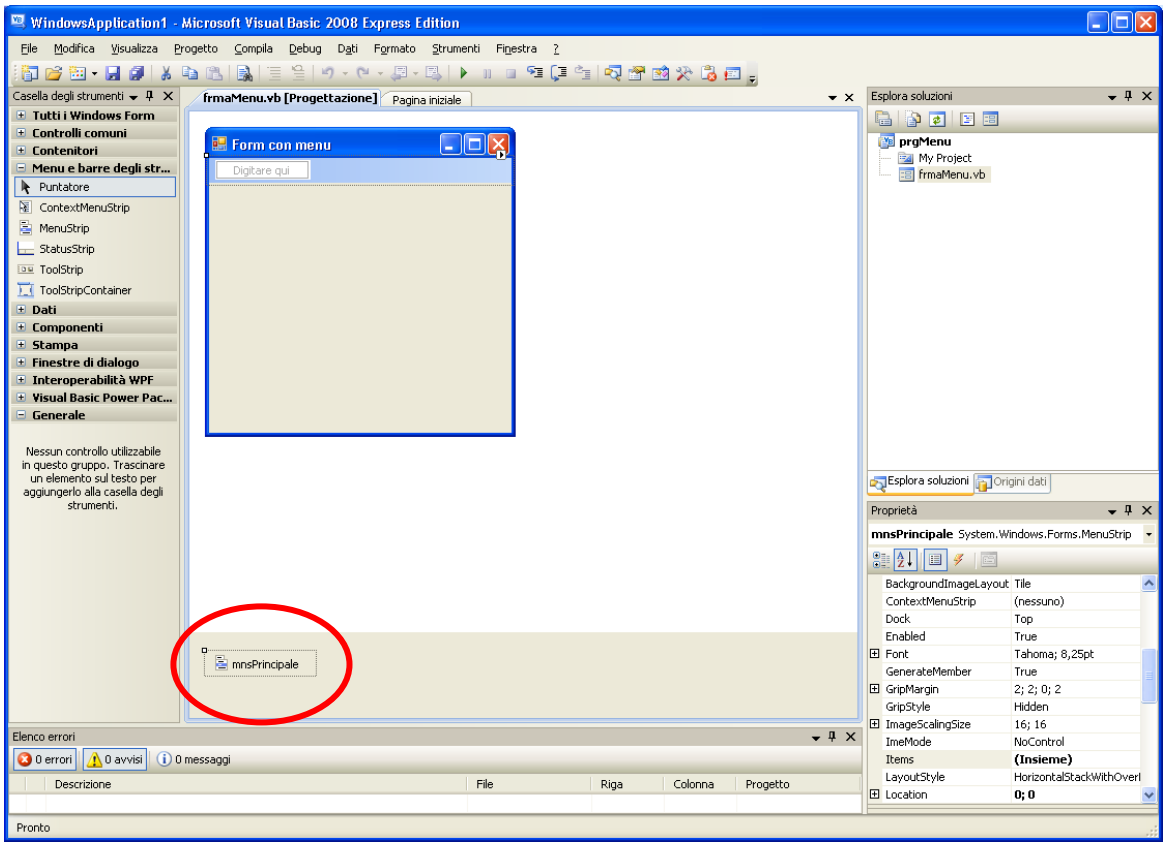


Cliccando sul pulsante **Aggiungi**, si aggiunge una nuova pagina.
Cliccando sul pulsante **Rimuovi**, si rimuove una pagina esistente.
Selezionando una pagina nell'elenco, è possibile impostarne le proprietà. La proprietà **Text** consente di scrivere il titolo della scheda.
Cliccare su OK per chiudere la finestra e applicare le impostazioni date.
Le proprietà di ogni scheda possono essere impostate anche nella comune finestra delle proprietà, dopo aver selezionato la scheda interessata.

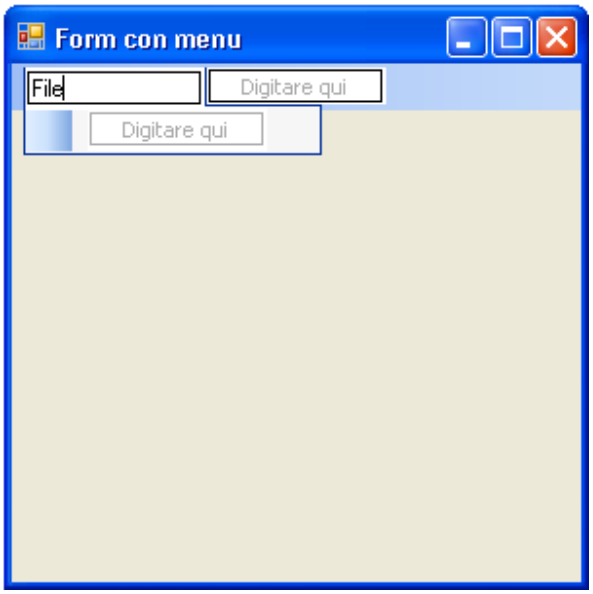
Gruppo menu e barre degli strumenti

MenuStrip

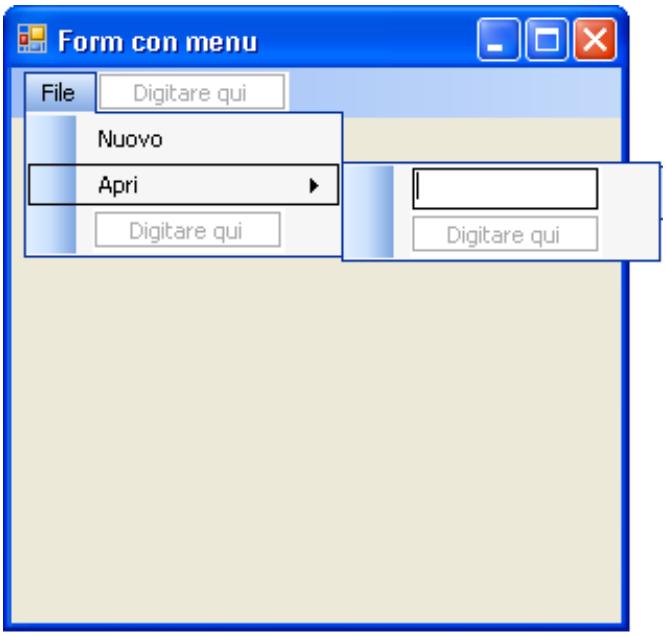
Il controllo MenuStrip permette di creare e modificare una barra dei menu sul Form.



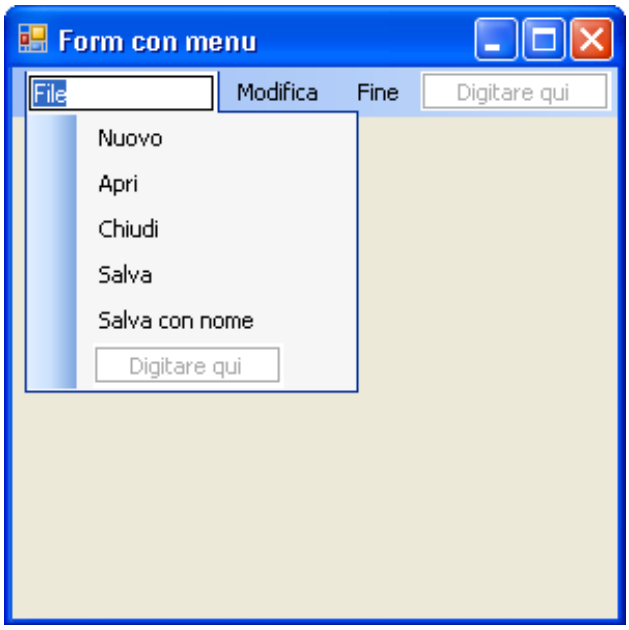
1. Per prima cosa, è necessario collocare il controllo sul Form. Appare, nella parte alta del Form, una striscia di colore celeste contenente una casella di testo bianca con dentro scritto *Digitare qui*. Dare un nome all'oggetto MenuStrip, adottando, come al solito, un criterio; ad esempio, secondo la notazione ungherese, far iniziare il nome con un prefisso come **mns** e completare il nome con una stringa descrittiva. Nella parte bassa dell'IDE appare un'immagine del controllo inserito.



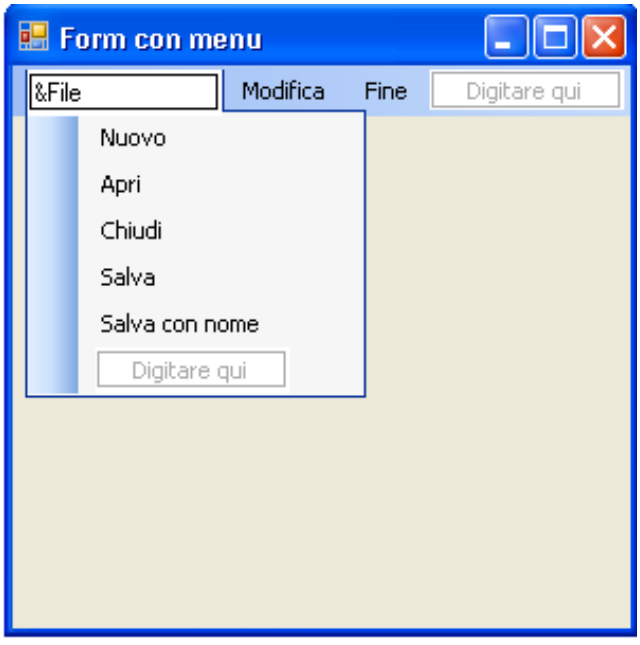
2. Cliccare nella casella con dentro scritto *Digitare qui* e digitare la prima opzione del menu. Appaiono anche una casella sotto e una alla sua destra, la prima per consentire l'inserimento della prima opzione di una tendina da associare all'opzione e la seconda per passare alla successiva opzione della barra dei menu.

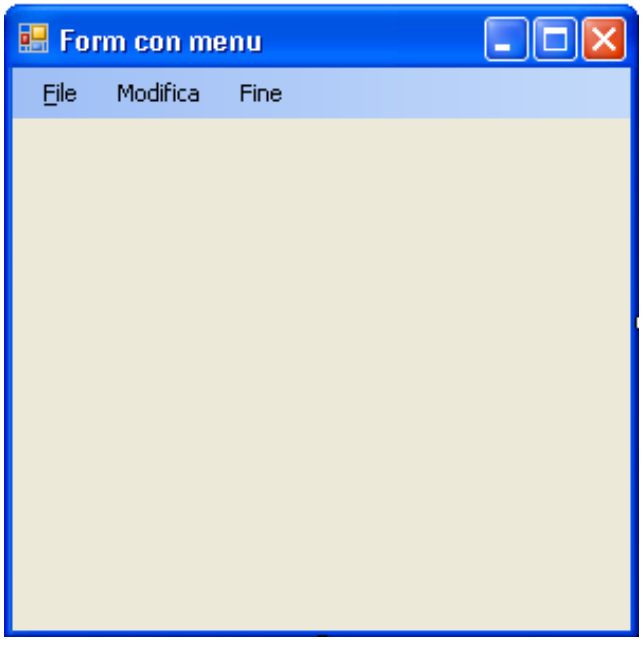


3. Per ogni voce che si digita, compaiono sempre una casella a fianco, per un sottomenu relativo a quella voce, e una casella sotto, per le successive opzioni della tendina.

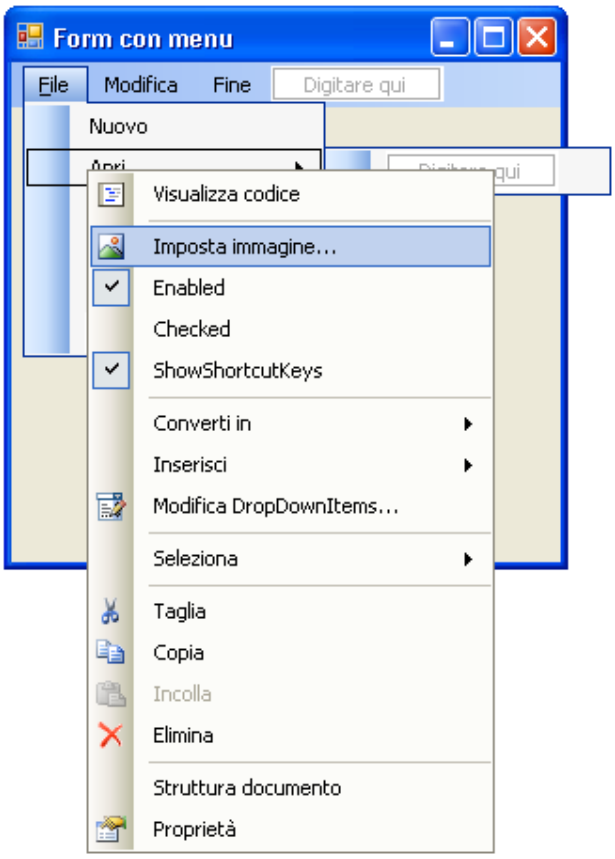


4. Per modifica un'opzione basta cliccare su di essa. Ricompare la casella che la contiene, in modo che si possa effettuare la modifica.

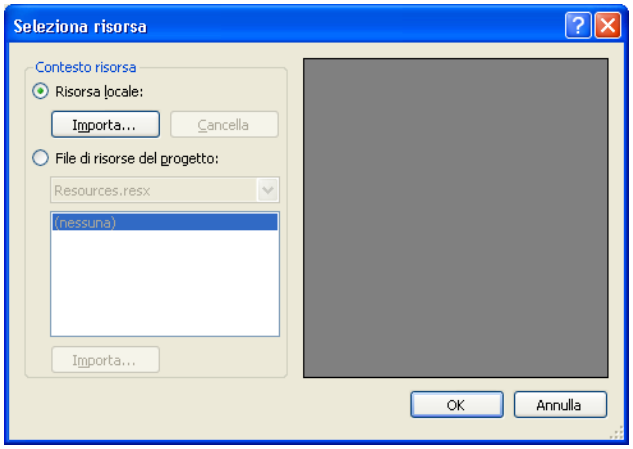


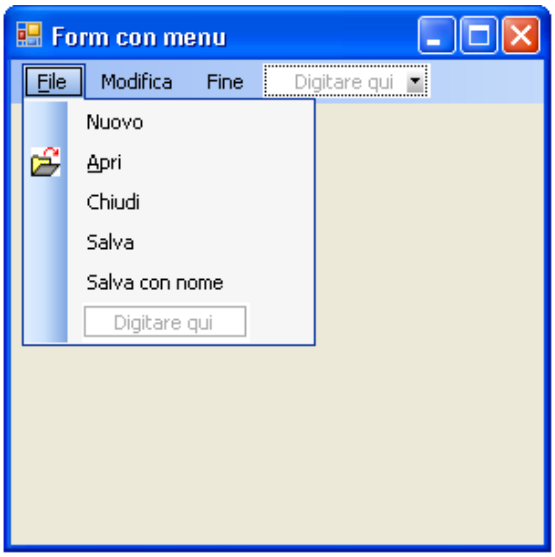


5. Scrivendo il carattere & davanti ad una lettera (anche interna) del nome di un'opzione del menu, essa apparirà sottolineata e sarà designata come **tasto di scelta** per la voce di menu. In tal modo, premendo, sulla tastiera, il tasto ALT e la lettera sottolineata, l'opzione sarà selezionata senza far ricorso al mouse.

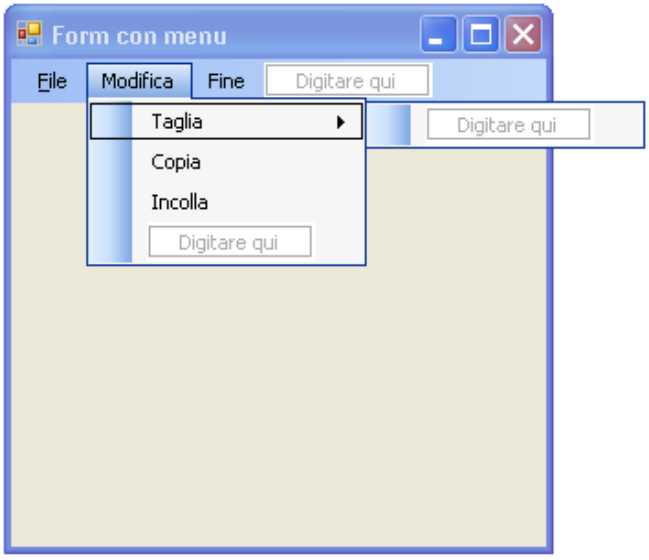


6. Cliccando con il tasto destro del mouse su una voce del menu in costruzione, si può selezionare l'opzione **Imposta immagine...**, per associare un'immagine a quella voce. Si può anche cliccare sulla proprietà **Image** nella finestra delle proprietà e cliccare sul pulsantino … alla sua destra.

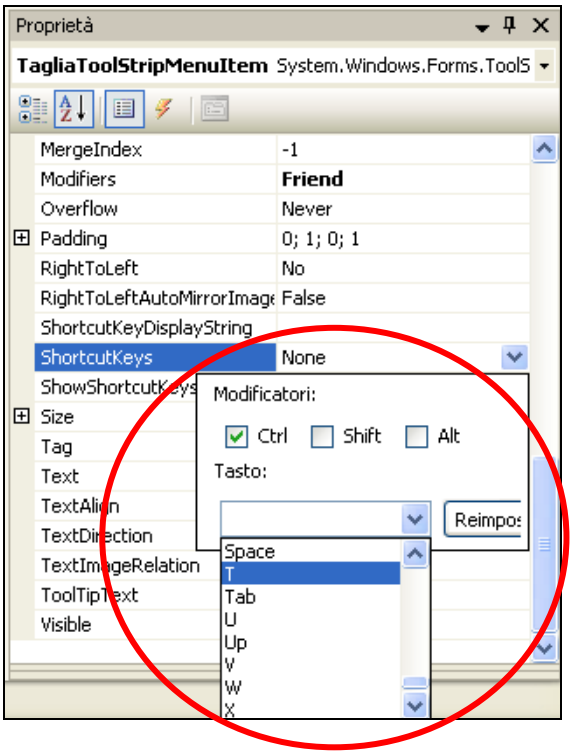


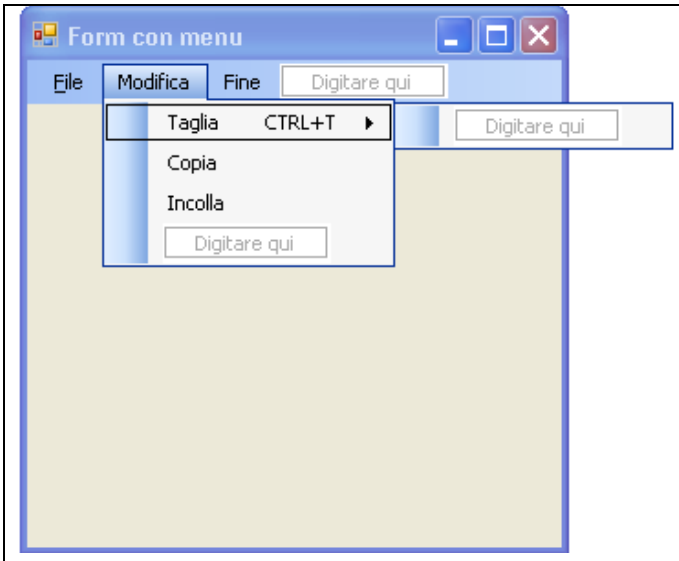


7. Dalla finestra **Seleziona risorsa** si può importare un'immagine dalle risorse del computer o scegliendola tra quelle già incluse nel progetto.



8. Selezionando un'opzione del menu in costruzione, si può, nella finestra delle proprietà, associare ad essa uno shortcut, vale a dire una combinazione di tasti. La proprietà da impostare è **ShortcutKeys**: compare una tendina per l'impostazione guidata.



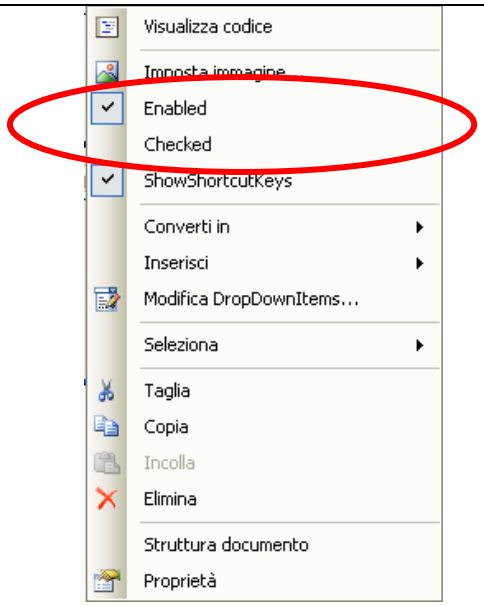


9. Premendo i tasti dello shortcut, l'opzione viene selezionata direttamente, senza passare per il menu.
La proprietà **ShowShortcutsKeys**, associata ad ogni opzione, se impostata a *False* rende l'eventuale shortcut dell'opzione invisibile.

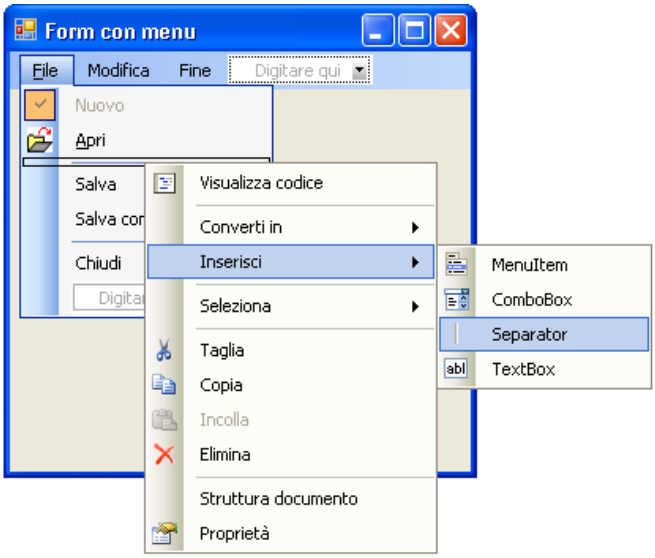
10. Per ogni opzione del menu è possibile impostare, nella finestra delle proprietà, anche le seguenti proprietà:

- **Checked:** consente di selezionare (Checked = *True*) o deselezionare (Checked = *False*) in modo stabile un'opzione (se è selezionata, appare un segno di spunta alla sua sinistra).
- **Enabled:** consente di abilitare (Enabled = *True*) o disabilitare (Enabled = *False*) un'opzione (se è disabilitata, l'opzione appare ingrigita).
- **Visibile:** consente di rendere un'opzione visibile (Visible = *True*) o invisibile (Visible = *False*).

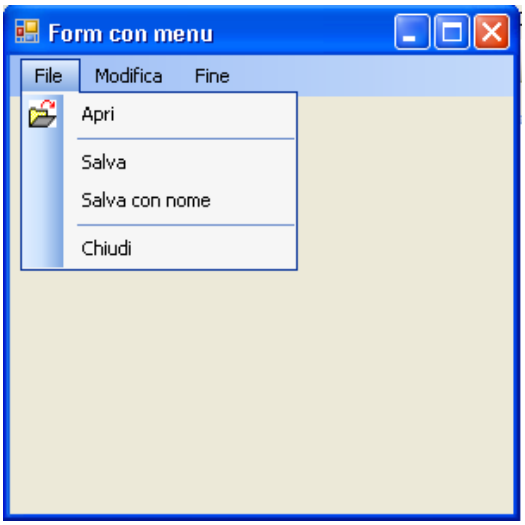
Le prime due proprietà possono anche essere impostate attraverso il menu contestuale che appare cliccando con il destro sull'icona del menu in fase di progettazione.



11. Con il mouse si può trascinare un'opzione, spostandola in alto o in basso all'interno di una tendina, o a sinistra o a destra nel menu principale.



12. Cliccando con il destro tra due opzioni e selezionando **Inserisci** – **Separator** dai menu contestuali che compaiono, si può inserire una linea di separazione tra le due opzioni.



13. Per ogni titolo di un menu può essere programmato l'evento click. Più precisamente,
    - se al titolo non corrisponde un sottomenu, si programma l'evento click tramite codice;
    - se al titolo corrisponde un sottomenu, cliccando su di esso comparirà automaticamente il sottomenu; è possibile in questo caso programmare l'evento click per ogni opzione del sottomenu, ad eccezione di quelle a cui corrispondono altri sottomenu, e così via.

Per la programmazione, cliccare due volte sull'opzione di interesse per far aprire la finestra del codice con l'intestazione e la chiusura del sottoprogramma corrispondente.
Ad esempio, cliccando sull'opzione *Apri* delle immagini di esempio, compare:

```
Private Sub ApriToolStripMenuItem_Click(...) Handles ApriToolStripMenuItem.Click

End Sub
```

Si osservi che, ad ogni opzione, viene attribuito un nome di default, che si può comunque modificare nella finestra delle proprietà. Il nome è costruito aggiungendo al titolo dell'opzione nel menu (nell'esempio, Apri) la stringa **ToolStripMenuItem**. Se si decide di assegnare ad ogni opzione un nome diverso, è sempre opportuno adottare un criterio, ad esempio, secondo la notazione ungherese, far iniziare ogni opzione con un prefisso come **mnu** e completare il nome con il titolo dell'opzione (ad esempio, mnuApri).

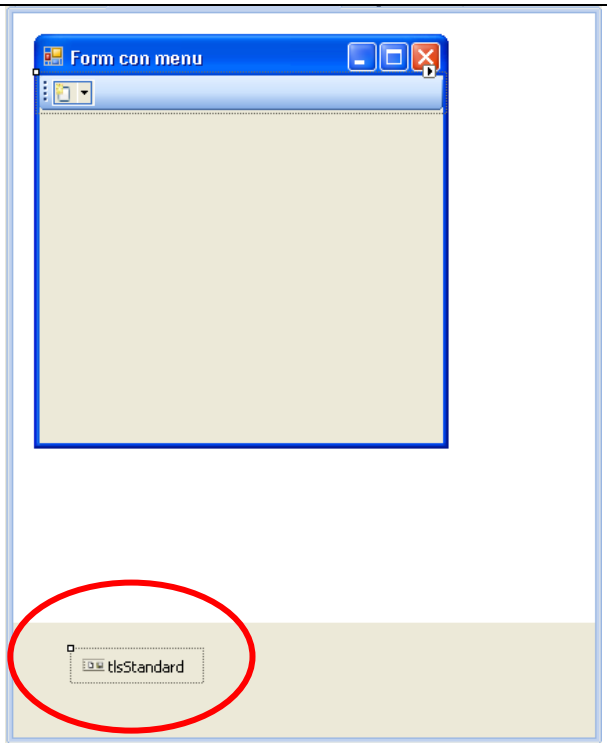
Tra Private Sub e End Sub, si scrivono, come già visto per gli altri oggetti, le istruzioni che devono essere eseguite se si clicca sull'opzione. Ad esempio, il sottoprogramma per il click sull'opzione Fine può essere il seguente:

```
Private Sub mnuFine_Click(...) Handles mnuFine.Click
    End
End Sub
```

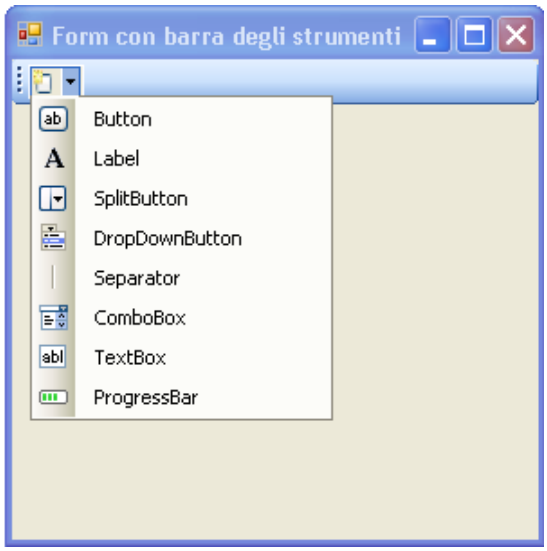
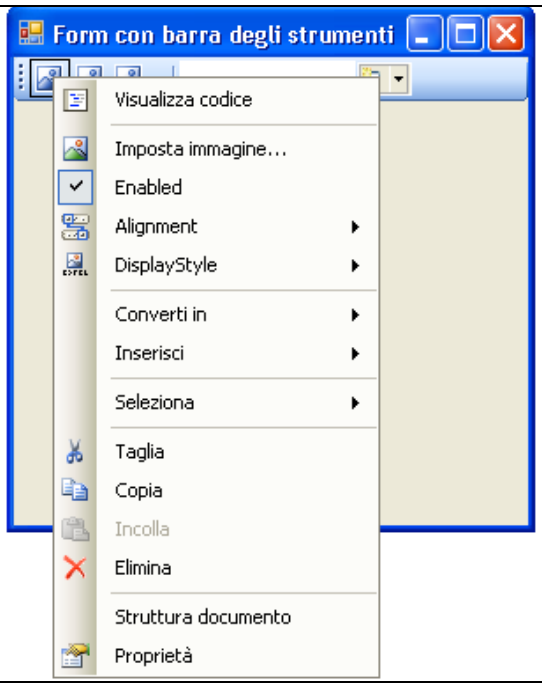
N.B.: L'istruzione **End** arresta l'esecuzione del programma immediatamente.

ToolStrip

Una ToolStrip è una barra di pulsanti (barra degli strumenti), che in genere viene collocata sotto il menu principale di una finestra. La sua costruzione avviene in modo simile a quanto visto per il controllo MenuStrip.



1. Collocato il controllo sul Form, appare, nella parte alta del Form, una striscia di colore celeste contenente uno splitButton. Dare un nome all'oggetto ToolStrip; il prefisso del nome può essere, in questo caso, **tls**. Nella parte bassa dell'IDE appare un'immagine del controllo ToolStrip inserito.

<table>
<tr>
<td> 
</td>
<td>2. Cliccare sul pulsantino dello SplitButton. Appare un menu a discesa che elenca gli strumenti che possono essere posizionati sulla barra.</td>
</tr>
<tr>
<td>
</td>
<td> 
</td>
</tr>
<tr>
<td>3. Gli oggetti selezionati dalla tendina compaiono in sequenza sulla barra. Ogni oggetto ha le sue proprietà, impostabili nella finestra delle proprietà.</td>
<td>4. Alcune proprietà possono essere impostate anche attraverso il menu contestuale che appare cliccando con il tasto destro del mouse sull'oggetto.</td>
</tr>
<tr>
<td>
</td>
<td>5. Ad esempio, attraverso l'opzione Imposta immagine... si può modificare l'immagine su un pulsante, con le stesse modalità già viste per la barra dei menu.
Le proprietà dipendono ovviamente dal tipo di oggetto.</td>
</tr>
</table>

6. Per quasi tutti gli oggetti è possibile impostare, nella finestra delle proprietà, le seguenti proprietà:
- **Enabled:** consente di abilitare (Enabled = *True*) o disabilitare (Enabled = *False*) un oggetto (se è disabilitata, l'oggetto appare ingrigito).
- **Visibile:** consente di rendere un oggetto visibile (Visible = *True*) o invisibile (Visible = *False*).

7. Con il mouse si può trascinare un oggetto, spostandolo a sinistra o a destra nella barra.

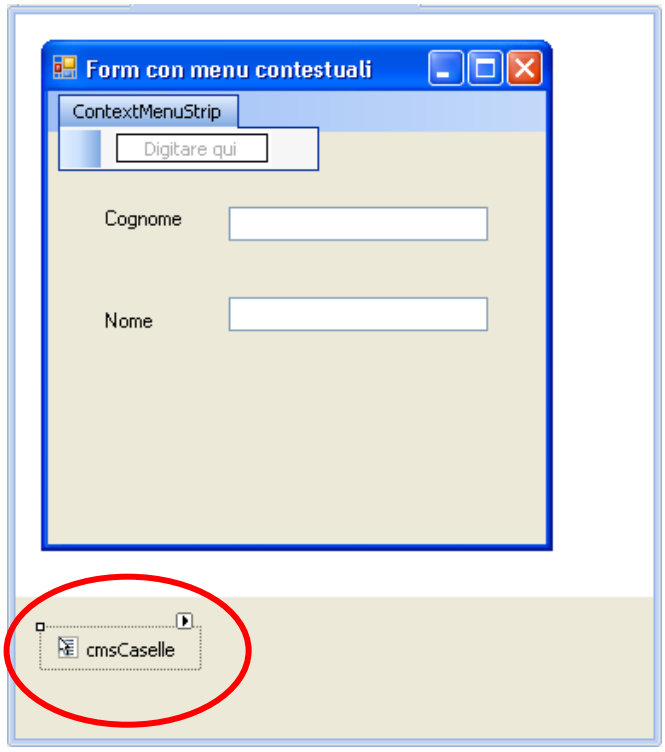
8. Per ogni oggetto della barra può essere programmato l'evento click o qualche altro evento (gli eventi dipendono dal tipo di oggetto).
Per la programmazione, cliccare due volte sull'opzione di interesse per far aprire la finestra del codice con l'intestazione e la chiusura del sottoprogramma corrispondente all'evento click.
Ad esempio, cliccando sul primo oggetto dell'esempio, che è un pulsante, compare:

```
Private Sub ToolStripButton1_Click(...) Handles ToolStripButton1.Click

End Sub
```

Si osservi che ad ogni oggetto viene attribuito un nome di default, che si può comunque modificare nella finestra delle proprietà. Il nome è costruito aggiungendo alla stringa **ToolStrip** il nome del tipo di oggetto e del suo numero d'ordine nella sequenza di creazione degli oggetti di quel tipo (`ToolStripButton1 al primo pulsante, ToolStripTextBox1 alla prima TextBox, ecc.`)
Se si decide di assegnare agli oggetti un nome diverso secondo la notazione ungherese, si può adottare, come prefisso, **tso (ToolStrip Object)** e completare il nome con una stringa che ne descriva le funzionalità.

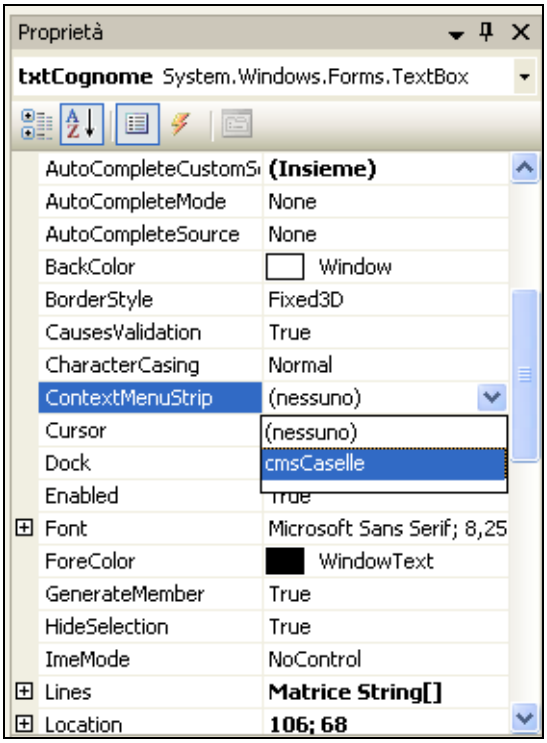
ContextMenuStrip

Permette di creare menu contestuali da associare agli oggetti del form o al form stesso, quando si clicca su di essi con il pulsante destro del mouse. Più oggetti possono condividere lo stesso menu contestuale.

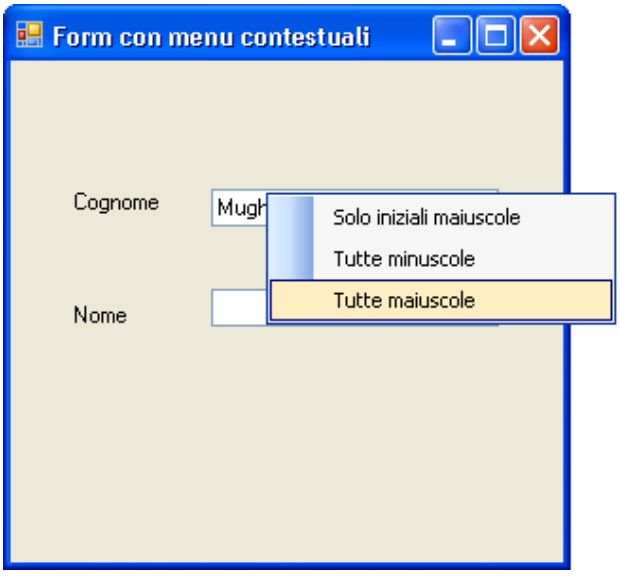


1. Per creare un menu contestuale, collocare un controllo ContextMenuStrip sul Form. Appare, nella parte alta del Form, una striscia di colore celeste contenente una casella di testo bianca con dentro scritto *Digitare qui*. Dare un nome all'oggetto; il prefisso del nome può essere, in questo caso, **cms**.
A differenza dei controlli MenuStrip e ToolStrip, un oggetto ContextMenuStrip scompare dal Form se si passa a lavorare su altri controlli in esso contenuti. Per farlo ricomparire, in modo da poterne modificare proprietà e contenuti, si deve cliccare due volte sull'immagine dell'oggetto che appare, come per gli altri due controlli, nella parte bassa dell'IDE quando l'oggetto viene creato.

2. Per la costruzione del menu vale tutto quanto detto per il controllo MenuStrip, anche per quanto riguarda il nome attribuito ad ogni opzione (se lo si vuole modificare, si può usare, come prefisso, **cmo**, che sta per Context Menu Option).
Nell'esempio, viene creato un menu contestuale per le caselle di testo, con tre opzioni: *Tutto maiuscole*, *Tutto minuscole*, *Solo iniziali maiuscole*.



3. Associare il menu creato all'oggetto interessato, modificando la proprietà **ContextMenuStrip** dell'oggetto.
Nell'esempio, questa operazione viene fatta per la prima casella di testo. Lo stesso può essere fatto per la seconda casella, a cui si può associare il medesimo menu contestuale dell'altra.

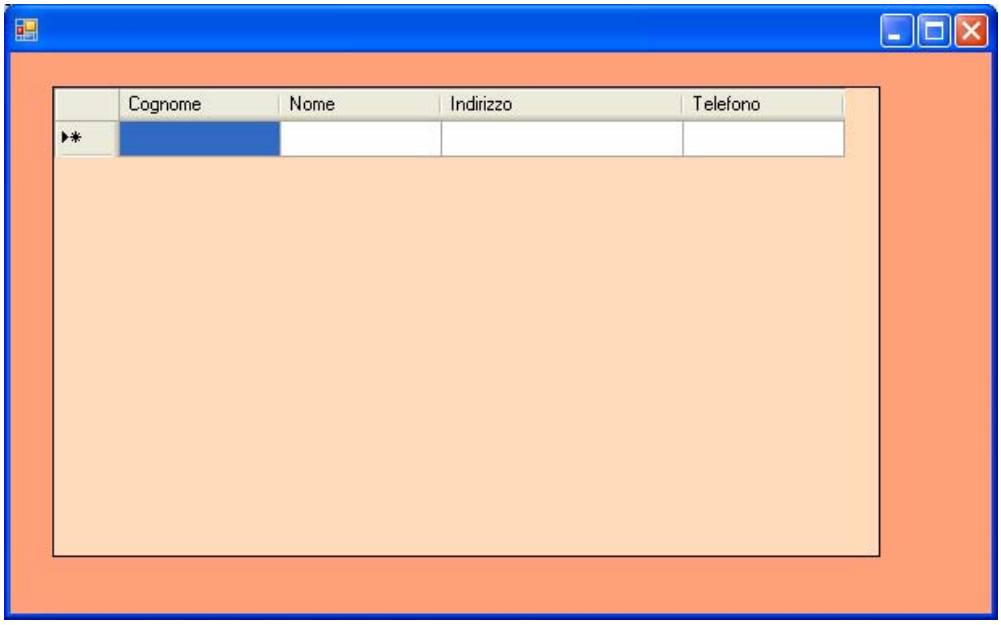


4. Durante l'esecuzione, cliccando con il pulsante destro del mouse sull'oggetto appare il menu contestuale.
Per la scrittura del codice corrispondente ad ogni opzione, vale sempre quanto detto per il controllo MenuStrip.

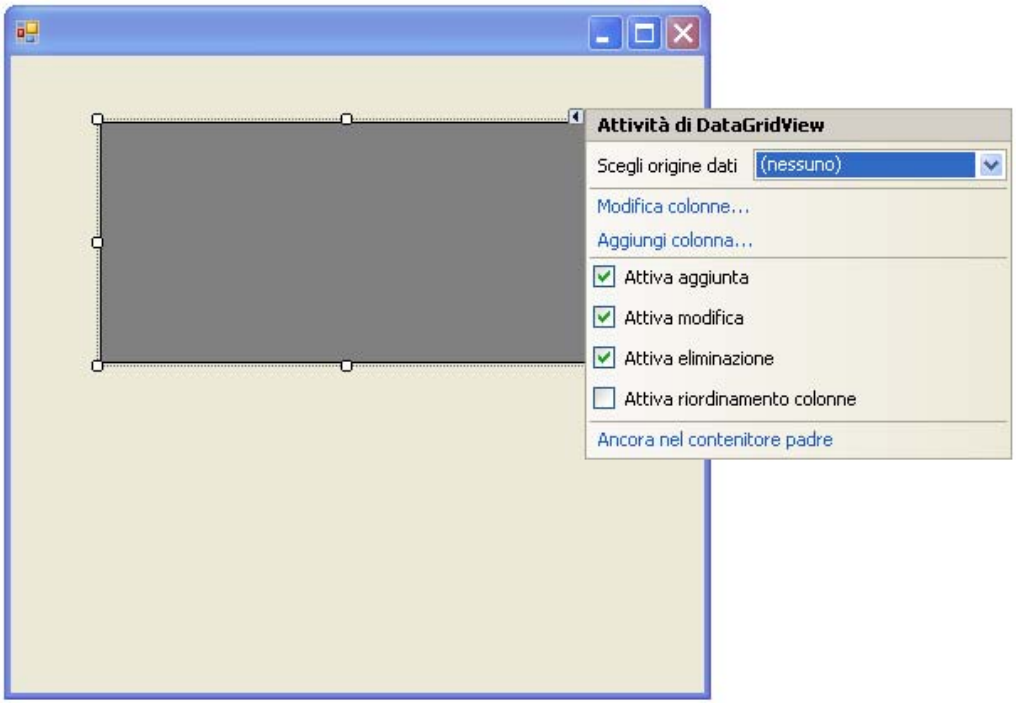
Gruppo Dati

DataGridView

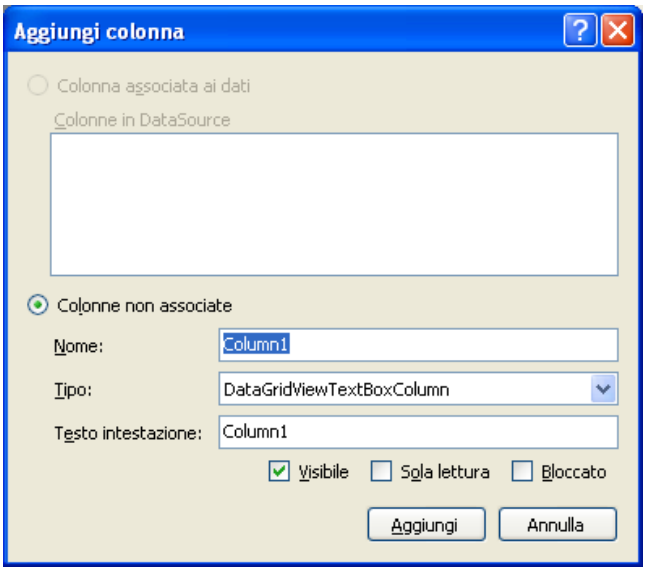
DataGridView è uno dei controlli più potenti del Framework .NET. Consente la visualizzazione di dati in una tabella, visualizzando una griglia di righe e colonne. L'intersezione tra una riga e una colonna costituisce una cella. In ciascuna cella è possibile inserire un testo o un'immagine.



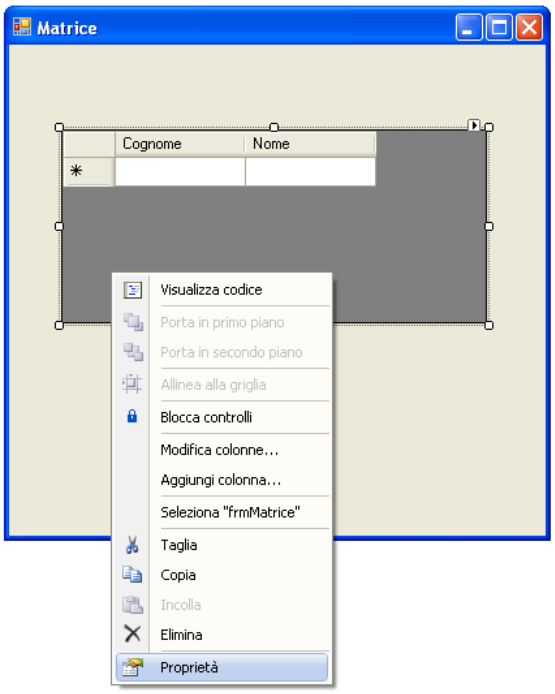
Quando l'oggetto viene posizionato sul form, appare un menu contestuale che dà all'utente la possibilità di personalizzarlo.

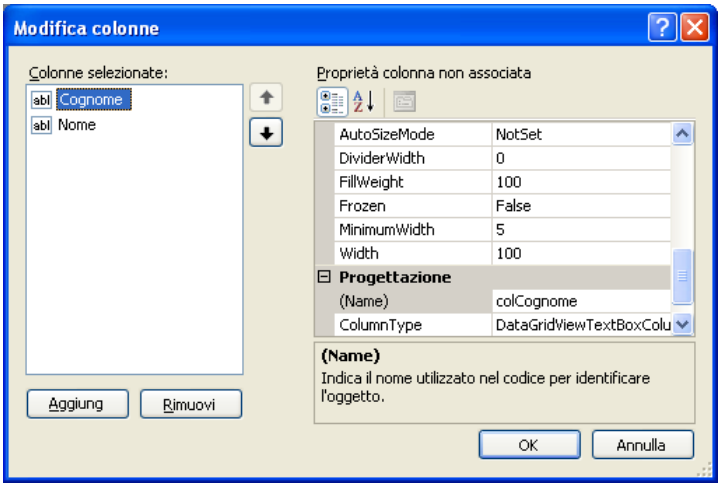


Selezionando l'opzione **Aggiungi colonna...**, in particolare, si ha la possibilità di comporre la tabella colonna per colonna, aggiungendo e personalizzando le colonne desiderate. Compare infatti la seguente finestra:



Ad ogni colonna si deve dare un nome e un testo da attribuire all'intestazione di colonna. Si può anche specificare se la colonna deve essere visibile, di sola lettura o modificabile, bloccata o no.
Una volta predisposta la tabella, si possono modificare le impostazioni date cliccando su di essa e selezionando **Modifica colonne...** .
Compare la finestra seguente:





Qui è possibile rimuovere e aggiungere colonne e impostare tutta una serie di proprietà.

Il numero di righe e il numero di colonne sono automaticamente assegnati alle proprietà **RowCount** (righe) e **ColumnCount** (colonne), che non possono essere impostate in fase di progettazione (non figurano nella finestra delle proprietà), ma solo in fase di esecuzione. In tali numeri non sono inclusi la riga e la colonna d'intestazione.
È possibile eliminare le intestazioni, lasciando solo le celle bianche. Impostare a tal fine al valore False le proprietà **ColumnHeadersVisible** e **RowHeadersVisible**. Si può anche scegliere di lasciare solo le intestazioni di colonna o solo quelle di riga.

Le righe e le colonne delle celle bianche sono numerate a partire da 0.

La proprietà **CurrentCell** contiene una serie di informazioni sulla cella corrente, che può essere impostata in fase di esecuzione, utilizzando il mouse o i tasti di direzione:

- le sottoproprietà **RowIndex** (CurrentCell.RowIndex) e **ColumnIndex** (CurrentCell.ColumnIndex) individuano la riga e la colonna della cella corrente.
- la sottoproprietà **Value** (CurrentCell.Value) rappresenta il valore contenuto nella cella corrente.

La proprietà **Item**, seguita da una coppia di coordinate (colonna, riga), fa riferimento alla cella con quelle coordinate. La sottoproprietà **Value** ne individua il contenuto.

```
Esempio:
      dgvTabella.Item(1,2).Value = 5

      scrive il valore 5 nella cella di riga 2 e colonna 1 della DataGridView di
      nome dgvTabella
```

Per scrivere nelle celle delle intestazioni di colonna si usa la proprietà **Columns**, che individua l'insieme di tutte le colonne del controllo. Se c è il numero di una colonna,

NomeDataGridView.Colums(c).HeaderCell.Value = testo

permette di scrivere un testo nell'intestazione della colonna c del DataGridView.
Per scrivere nelle celle delle intestazioni di riga si usa, analogamente, la proprietà **Rows**, che individua l'insieme di tutte le righe del controllo. Se r è il numero di una riga,

NomeDataGridView.Rows(r).HeaderCell.Value = testo

permette di scrivere un testo nell'intestazione della riga r del DataGridView.

Esempi:

- `dgvTabella.Columns(0).HeaderCell.Value = "Cognome"`
  `scrive la parola "Cognome" come intestazione della colonna 0.`
- `dgvTabella.Rows(nr).HeaderCell.Value = nr`
  `scrive come intestazione della riga nr lo stesso numero di riga.`

Per definire la larghezza di una colonna, si usa

NomeDataGridView.Colums(c).Width = larghezza

dove la larghezza è espressa in **pixel** (punti dello schermo).

Per definire l'altezza di una riga, si usa

NomeDataGridView.Rows(r).Height = altezza

Anche l'altezza è espressa in **pixel**.

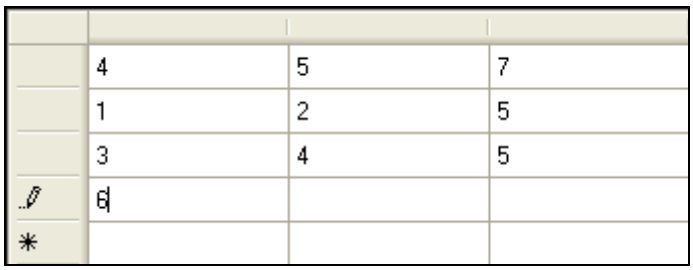
Se il testo di una cella non è visualizzato interamente, si può intervenire sulla proprietà **AutoSizeColumnsMode** del controllo; essa determina in che modo le colonne vengano ridimensionate quando il testo contenuto va oltre i confini visibili. I valori che può assumere sono:

- *None*: le colonne rimangono sempre della stessa larghezza, a meno che l'utente non le ridimensioni manualmente;
- *AllCells*: la colonna viene allargata affinché il testo di *tutte* le celle sottostanti e della stessa intestazione sia visibile;
- *AllCellsExceptHeader*: la colonna viene allargata in modo che solo il testo di tutte le celle sottostanti sia visibile;
- *ColumnHeader*: la colonna viene allargata in modo che il testo dell'header sia interamente visibile;
- *DisplayedCells*: come AllCells, ma solo per le celle visibili nei margini del controllo;
- *DisplayedCellsExceptHeader*: come AllCellsExceptHeader, ma solo per le celle visibili nei margini del controllo;
- *Fill*: le colonne vengono ridimensionate affinché la loro larghezza totale sia quanto più possibile vicina all'area effettivamente visibile a schermo, nei limiti imposti dalla proprietà **MinimumWidth** di ciascuna colonna.

La proprietà **AutoSizeRowsMode** è l'analogo per le righe.

Altre utili proprietà sono:

AllowUserToAddRows: indica se l'utente può (True) o non può (False) aggiungere righe. Se l'utente può aggiungere righe, quando inizia a digitare dati nell'ultima riga disponibile, compare già una nuova riga etichettata con *. Se l'utente non può aggiungere righe, questo non accade.



Enabled: abilita/disabilita (True/False) la possibilità, per l'utente, di modificare il contenuto della DataGridView.

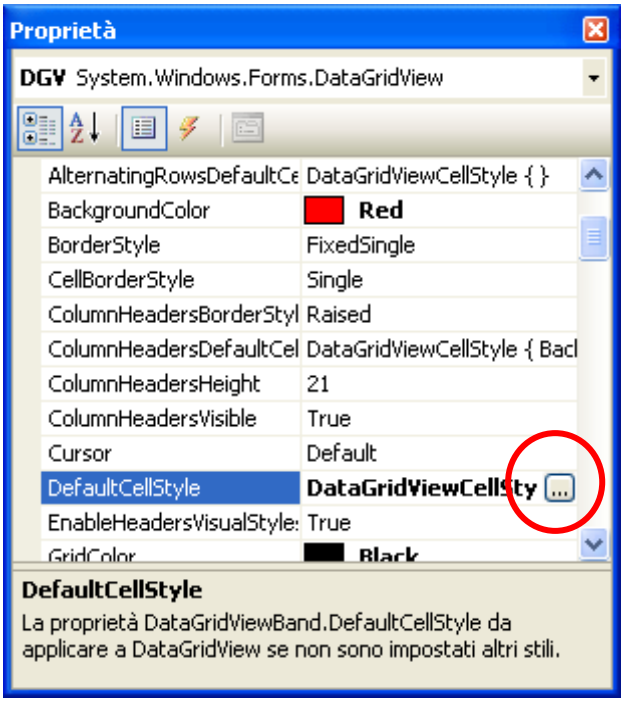
BackgroundColor: imposta il colore dello sfondo su cui poggia la griglia, limitatamente alle celle non fisse. Il colore si assegna scrivendo **Color.NomeColore**

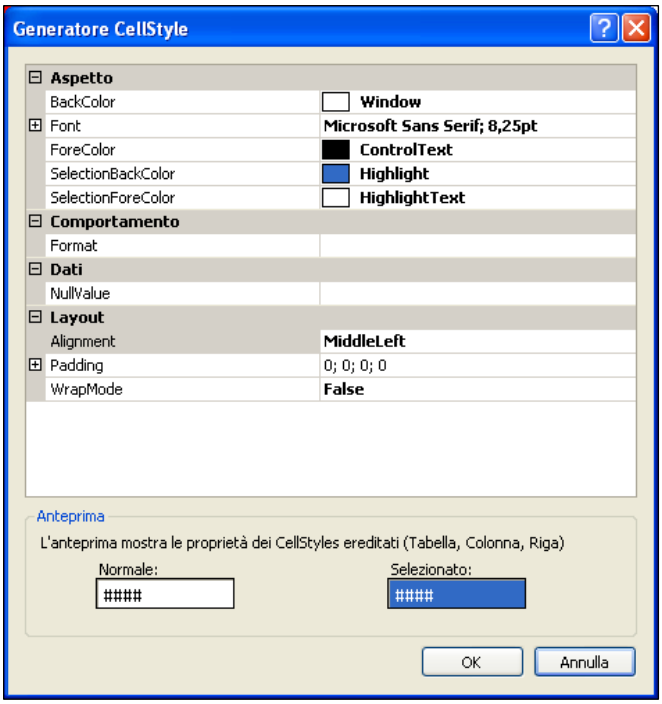
```
Es.: dgvTabella.BackColor = Color.Red   colora di rosso le celle non fisse.
```

Si può impostare anche nella finestra delle proprietà, selezionando il colore dalla tavolozza.

CellBorderStyle: definisce il bordo delle celle.

DefaultCellStyle: imposta varie caratteristiche delle celle, quali il colore delle celle (**BackColor**), il font (**Font**), il colore dei caratteri (**ForeColor**), l'allineamento del contenuto delle celle (**Alignment**), ecc. Se si opera nella finestra delle proprietà, occorre cliccare sul pulsante con tre puntini che appare a destra del nome della proprietà. Comparirà una finestra in cui è possibile effettuare tutte le impostazioni.





Per impostare le proprietà da codice, la sintassi è:

NomeDataGridView.DefaultCellStyle.NomeProprietà = valore

Esempio:

```
dgvTabella.DefaultCellStyle.BackColor = Color.Yellow
colora le celle di giallo
```

ColumnHeadersDefaultCellStyle/**RowHeadersDefaultCellStyle**: impostano in modo analogo le caratteristiche delle celle delle intestazioni.

ColumnHeadersHeight /**RowHeadersWidth** : impostano le dimensioni delle intestazioni (altezza della riga delle intestazioni delle colonne, larghezza della colonna delle intestazioni delle righe).

ColumnHeadersBorderStyle/**RowHeadersBorderStyle** : impostano lo stile dei bordi delle celle delle intestazioni.

Alcune impostazioni sulle celle di intestazione di righe e colonne richiedono che siano disattivati gli stili impostati nel sistema operativo. Impostare, a Tal fine, la proprietà **EnabledHeadersDefaultStyle** della DataGridView a **False**, altrimenti le impostazioni date saranno ignorate.

Infine, qualche utile metodo.

Add() consente di aggiungere una riga

NomeDataGridView.Rows.Add()

o una colonna

NomeDataGridView.Column.Add(nome, intestazione)

Per aggiungere una colonna, occorre specificare, come parametri di Add, il nome e l'intestazione da dare alla colonna (due parametri di tipo stringa).

Il metodo **RemoveAt()** permette invece di eliminare una riga di cui si specifica il numero

NomeDataGridView.Rows.RemoveAt(r)

o una colonna di cui si specifica il numero

NomeDataGridView.Columns.RemoveAt(c)

Esempio:

```
dgvMatrice.Columns.RemoveAt(dgvMatrice.CurrentCell.ColumnIndex)
elimina la colonna di cui fa parte la cella corrente.
```

Per svuotare la tabella del suo contenuto, si usano le istruzioni:

NomeDataGridView.Rows.Clear()
NomeDataGridView.Columns.Clear()

Variabili semplici e strutturate

Le variabili possono essere
- semplici (o scalari)
- strutturate

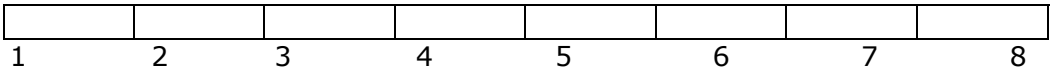
Le variabili del primo tipo sono in grado di ospitare un solo valore per volta.

Le variabili del secondo tipo sono invece vere e proprie strutture in grado di ospitare contemporaneamente più valori. Le variabili strutturate sono essenzialmente di due tipi:
- array
- record

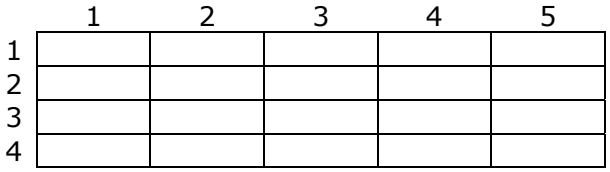
Gli array

In un array tutti i valori, denominati **elementi**, hanno lo stesso tipo ed occupano opportuni spazi numerati attraverso numeri d'ordine denominati **indici**.

Un array può avere più indici. Si parla di
- **vettore o array monodimensionale**, se l'indice è uno solo. La struttura corrispondente può essere così schematizzata:



- **matrice o array bidimensionale**, se gli indici sono due, uno che individua la riga e l'altro che individua la colonna. La struttura corrispondente può essere così schematizzata:



- **array multidimensionale**, se gli indici sono più di 2 (fino ad un massimo di 60). In questo caso si riesce a raffigurare la struttura solo se le dimensioni sono 3 (un cubo),

Come tutte le variabili, anche l'array ha un **nome**, che indica l'intera struttura. Un singolo elemento è invece indicato dal nome dell'array seguito dall'indice (o dagli indici) tra parentesi tonde.

I vettori

Quanti valori può accogliere un vettore? Dipende da quanti ne sono stati indicati nell'apposita istruzione di dichiarazione, che può essere eseguita per mezzo dell'istruzione **Dim**, indicando il nome del vettore, il massimo valore dell'indice e il tipo degli elementi:

Dim nome_vettore (IndiceMassimo) As tipo

Esempi:

```
Dim Nomi (20) As String
Dim Numeri (100) As Integer
Dim Dati (9) As Single
```

Il valore minimo dell'indice è, per assunzione, zero e questa impostazione non può essere cambiata. È comunque possibile, se si preferisce, utilizzare gli elementi a partire dalla posizione 1 o da un'altra posizione (in molti casi può essere comodo farlo).

Dato che, come abbiamo visto, gli indici sono sempre a base 0, il numero effettivo di elementi presenti nel vettore sarà di un'unità superiore rispetto all'indice massimo. Pertanto i vettori dichiarati nei precedenti esempi hanno rispettivamente 21, 101 e 10 elementi.
Si può anche dichiarare un vettore e, al tempo stesso, caricarlo. La sintassi è:

Dim nome_vettore (IndiceMassimo) As tipo = {elementi dell'array separati da virgole}

```
Esempio:
      Dim giorni() As String = {“Lunedì”, “Martedì”, “Mercoledì”, “Giovedì”, _
                                “Venerdì”, “Sabato”}
```

La dichiarazione con Dim obbliga il programmatore a specificare la dimensione massima dell’indice mediante dati costanti. Si parla in questo caso di **vettori statici o a dimensione fissa** e per utilizzarli è necessario già conoscere il numero di elementi del vettore o sovradimensionarlo.
Molti programmi non utilizzano i vettori statici proprio perché raramente il programmatore conosce a priori il numero di elementi. In alternativa è allora possibile utilizzare i **vettori dinamici o a dimensione variabile**, per i quali la dimensione non deve essere obbligatoriamente nota a priori, ma può essere impostata in fase di esecuzione.

Per creare un vettore dinamico occorre:

- dichiarare il vettore con Dim senza indicare l’indice massimo, ma solo le parentesi aperta e chiusa () e il tipo.
  Esempio:
  ```
  Dim Numeri() As Integer
  ```

- dimensionare il vettore, quando necessario, quindi anche più volte all'interno dello stesso programma, utilizzando l'istruzione **ReDim**, secondo la sintassi:
  ReDim nome_vettore (IndiceMassimo)

ReDim rende possibile il dimensionamento del vettore attraverso l’utilizzo di variabili per indicare l’estremo superiore dell’indice.

```
Esempio:
                        ReDim Numeri (N)
```

Ogni ridimensionamento di un vettore con ReDim ne azzera completamente i dati, a meno che non si usi la clausola **Preserve**, come nell’esempio seguente:

```
                        ReDim Preserve Numeri (N + 10)  As Integer
```

dove si aggiungono 10 posizioni al vettore Numeri, senza perdere il contenuto delle prime N posizioni. In assenza di Preserve, il ridimensionamento dell'array comporterebbe la cancellazione di tutti i valori contenuti.

L’uso di Redim e Preserve ha però prestazioni molto scarse a causa dell'eccessiva lentezza. Entrambi derivano dal Visual Basic 6.0 e non fanno parte, pertanto, della sintassi .NET, sebbene continuino ad essere molto usate, sia per comodità, sia per abitudine. Il metodo più corretto da adottare consiste nell'usare la procedura **Array.Resize**.
La sintassi é:

```
                        Array.Resize(array, newSize)
```

dove:

- Array è il nome del vettore,
- newSize è la nuova dimensione del vettore (numero di elementi e non indice massimo, come in Dim e ReDim).

Questo metodo alloca un nuovo vettore della dimensione specificata, copia gli elementi del vettore vecchio nel vettore nuovo e quindi sostituisce il vettore vecchio con il vettore nuovo.

Caricamento di un vettore

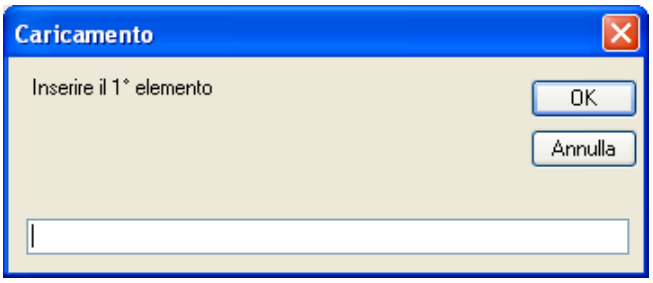
È la fase in cui il vettore viene riempito, ad esempio con dati immessi dall'utente. In questo caso il caricamento può essere di due tipi:
- determinato
- indeterminato

Il **caricamento determinato** dei dati si effettua quando si conosce il numero di elementi da inserire nel vettore. Può pertanto essere realizzato attraverso un ciclo, in particolare il ciclo For Next, in quanto l'indice deve variare dalla prima all'ultima posizione del vettore da riempire.
Per caricare tutti gli elementi del vettore in un ciclo, l'utente potrà inserire i dati in una serie di InputBox.

Il ciclo è allora del seguente tipo:

```
For K=1 To N Step 1
      V(K) = InputBox ("Inserire il " & K & "° elemento", "Caricamento")
Next
```

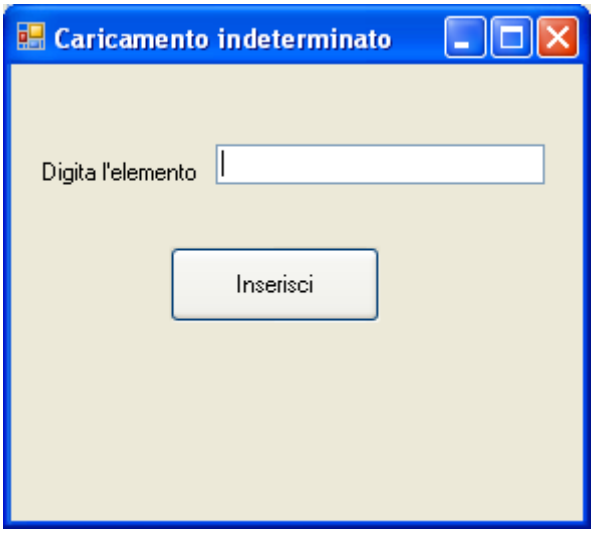
All'utente apparirà la seguente InputBox per ogni elemento del vettore (ciascuno con il proprio indice):



Se il vettore è numerico, occorrerà inserire la funzione InputBox all'interno della funzione Val.

Il caricamento determinato è, in realtà, poco funzionale, perché obbliga ad introdurre preventivamente il numero di elementi da inserire o, al più, ad arrestare l'inserimento attraverso un "tappo".

Il modo più efficiente di procedere consiste nel lasciare che l'utente inserisca gli elementi quando ritiene opportuno, senza doverne indicare a priori il numero. Si parla, in tal caso, di **caricamento indeterminato**.
Per l'inserimento di elementi nel vettore sarà sufficiente mettere a disposizione dell'utente:
- una casella di testo in cui digitare l'elemento da inserire,
- un pulsante su cui cliccare per prelevare il dato dalla casella di testo ed aggiungerlo nella posizione successiva all'ultimo, estendendo la dimensione del vettore (conservandone il precedente contenuto).



```
Dim V() As String
Dim N As Int16 = 0
Private Sub btnInserisci_Click(...)Handles btnInserisci.Click
   N = N + 1
   ReDim Preserve V(N)
   V(N) = txtDato.Text
   txtDato.Text = ""
   txtDato.Focus()
End Sub
```

Visualizzazione degli elementi contenuti in un vettore

Per visualizzare tutti i dati contenuti in un vettore è necessario utilizzare un ciclo. La visualizzazione potrà essere effettuata all'interno di un oggetto che consente l'elencazione di dati, come la ListBox.

Il ciclo potrebbe quindi essere il seguente:

```
Private Sub btnVisualizza _Click(...) Handles btnVisualizza.Click
    lstVettore.Items.Clear()  ' Svuota la lista per visualizzazioni successive
    For K = 1 To N Step 1
      lstVettore.Items.Add(V(K))
    Next
End Sub
```

Implementazione di una tabella attraverso vettori paralleli

Affiancando due o più vettori anche di tipo diverso ma con lo stesso numero di elementi, è possibile memorizzare in tali strutture i dati contenuti in una tabella; gli elementi con lo stesso indice dei tre vettori, insieme, costituiscono una riga della tabella.

Si parla in questo caso di vettori paralleli.

Esempio:

Per implementare una tabella di N righe e 3 colonne contenenti i cognomi, i nomi e le età di N persone, si possono utilizzare 3 vettori (uno per colonna) così dichiarati:

```
Dim Cognomi(N-1) As String
Dim Nomi(N-1) As String
Dim Eta(N-1) As Byte
```

} Per numerare le righe da 0 a N-1

Se si vuole numerare le righe da 1 a N, la dichiarazione sarà la seguente:

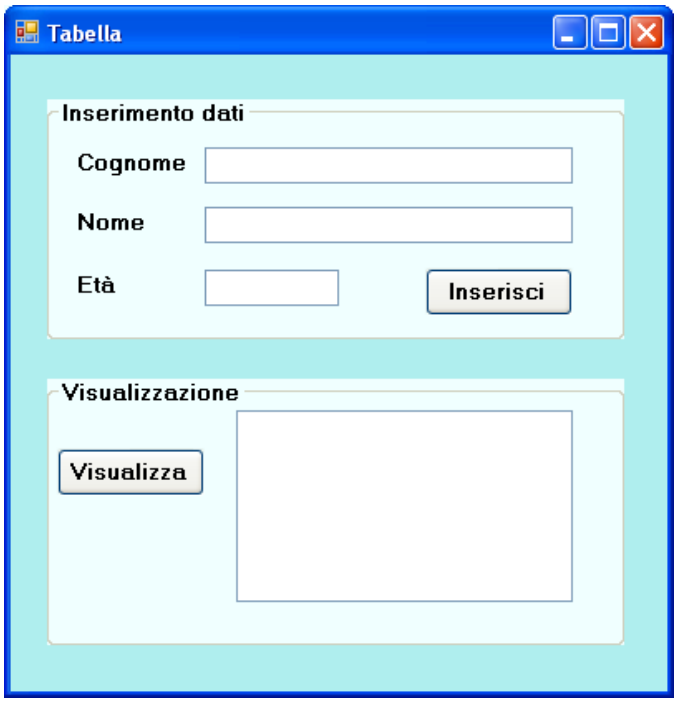
```
Dim Cognomi(N) As String
Dim Nomi(N) As String
Dim Eta(N) As Byte
```

Per l'inserimento e la visualizzazione dei dati il meccanismo è analogo a quello in cui il vettore è uno solo. Cambia ovviamente il numero di dati da acquisire o visualizzare ogni volta.
Per l'inserimento, ad esempio, le caselle di testo in cui inserire i dati prima di cliccare sul pulsante sarebbero tante quanti sono i vettori, mentre nella visualizzazione i dati dovrebbero essere concatenati, costituendo un'unica riga, prima di visualizzarli nella ListBox.

Es.:

Facendo riferimento ai vettori dell'esempio precedente, vediamo come è possibile inserire dati nei vettori attraverso il caricamento indeterminato e visualizzarli in una ListBox.

L'interfaccia potrebbe essere la seguente:

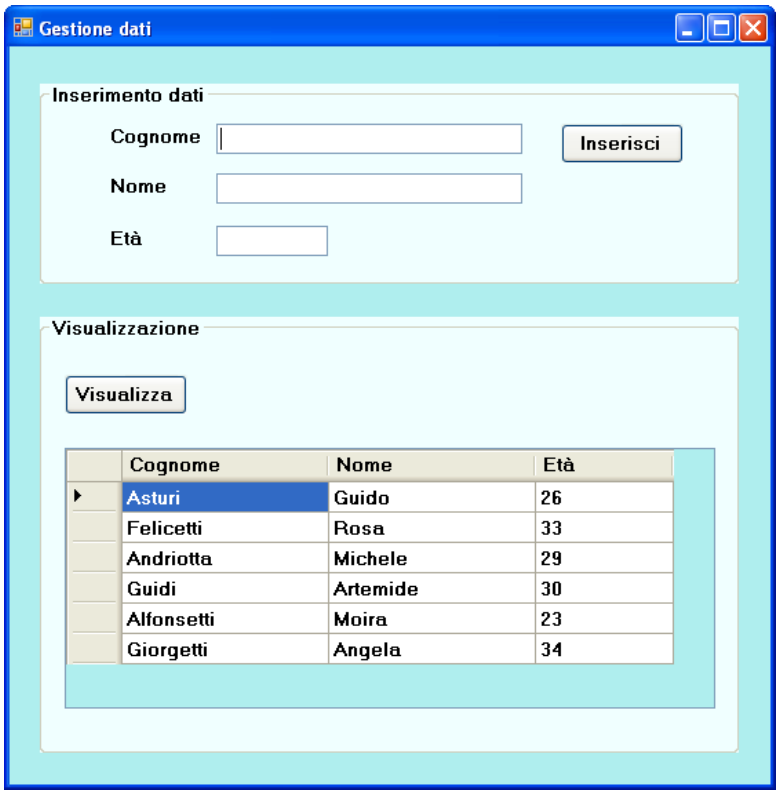



```
Public Class frmVettoriParalleli

  Dim Cognomi() As String
  Dim Nomi() As String
  Dim Eta() As Byte
  Dim N As Byte = 0

  Private Sub btnInserisci_Click(...) Handles btnInserisci.Click
    N = N + 1
    Redim Cognomi(N)
    Redim Nomi(N)
    Redim Eta(N)
    Cognomi(N) = txtCognome.Text
    Nomi(N) = txtNome.Text
    Eta(N) = Val(txtEta.Text)
    txtCognome.Clear()
    txtNome.Clear()
    txtEta.Clear()
    txtCognome.Focus()
  End Sub

  Private Sub btnVisualizza_Click(...) Handles btnVisualizza.Click
    Dim K As Byte
    lstElenco.Items.Clear()
    For K = 1 To N Step 1
      lstElenco.Items.Add(Cognomi(K) & vbTab & Nomi(K) & vbTab & Eta(K))
    Next
  End Sub

End Class
```

Per la visualizzazione di tabelle realizzate attraverso vettori paralleli o in altro modo, si può anche utilizzare il controllo DataGridView, sicuramente più idoneo rispetto alla ListBox.

Es.:

Facendo riferimento al programma dell'esempio precedente, vediamo come cambiano l'interfaccia e la routine di visualizzazione.



Rispetto al programma nell'esempio precedente, cambia solo la routine che gestisce l'evento click sul pulsante Visualizza:

```
Private Sub btnVisualizza_Click(...) Handles btnVisualizza.Click
 Dim K As Byte, nr As Byte
 nr = 1
 For K = 1 To N Step 1
    dgvElenco.RowCount = nr
    dgvElenco.Item(0, K - 1).Value = Cognomi(K)
    dgvElenco.Item(1, K - 1).Value = Nomi(K)
    dgvElenco.Item(2, K - 1).Value = Eta(K)
    nr = nr + 1
 Next
End Sub
```

Le matrici

Le **matrici**, o **array bidimensionali**, sono, come i vettori, variabili di **tipo strutturato**, ovvero variabili in grado di ospitare insiemi di valori tutti dello stesso tipo. Gli elementi della matrice sono però disposti in una struttura bidimensionale del seguente tipo:

	1	2	3	4	5
1					
2					
3					
4					

Come il vettore, anche la matrice ha un nome, che indica l'intera struttura. Un singolo elemento è invece indicato dal nome della matrice, seguito dall'indice di riga e dall'indice di colonna, tra parentesi tonde.

```
Es.: MAT(2,3) indica l'elemento della matrice di nome MAT in riga 2 e colonna 3.
```

Anche la dichiarazione della matrice può essere effettuata con **Dim**, indicando il nome della matrice, il massimo valore degli indici e il tipo degli elementi:

Dim nome (indice_max_riga, indice_max_colonna) As tipo

```
Es.: Dim MAT(4, 5)  As Single
```

Anche in questo caso il valore minimo dell'indice sarà, per assunzione, zero. Quindi la matrice così dichiarata

Dim MAT(4, 5) As Single

ha 5 righe (da 0 a 4) e 6 colonne (da 0 a 5).

Come per i vettori, comunque, non si è obbligati ad utilizzare l'indice 0.

Per creare una matrice dinamica occorre:

- dichiarare la matrice con Dim senza indicare l'indice massimo di ogni dimensione, ma solo le parentesi aperta e chiusa con una virgola all'interno (,) e il tipo.
  Esempio:
  ```
  Dim Numeri(,) As Integer
  ```

- dimensionare la matrice, quando necessario, quindi anche più volte all'interno dello stesso programma, utilizzando l'istruzione **ReDim**, secondo la sintassi:
  ReDim nome_matrice (indice_max_riga, indice_max_colonna)

ReDim rende possibile il dimensionamento del vettore attraverso l'utilizzo di variabili per indicare gli estremi superiori degli indici.

Esempio:
```
ReDim Numeri (NR, NC)
```

Ogni ridimensionamento di una matrice con ReDim ne azzera completamente i dati, a meno che non si usi la clausola **Preserve**; è possibile espandere però solo il numero di colonne e non il numero di righe, come nell'esempio seguente:

```
ReDim Preserve Numeri (NR, NC+10)  As Integer
```

dove si aggiungono 10 colonne alla matrice Numeri, senza perdere il contenuto delle prime NC colonne. In assenza di Preserve, il ridimensionamento dell'array comporterebbe la cancellazione di tutti i valori contenuti.

Caricamento di una matrice

È la fase in cui la matrice viene riempita, ad esempio con dati inseriti dall'utente. L'acquisizione dei dati avviene generalmente attraverso due cicli For Next nidificati, uno per le righe e uno per le colonne, in quanto l'indice deve variare, per ogni coordinata, dalla prima all'ultima posizione, anche se è possibile inserire gli elementi in ordine sparso.

Considerando una matrice di nome Mat, di M righe, numerate da 1 a M, e N colonne, numerate da 1 a N, il caricamento può avvenire

- per righe: gli elementi vengono inseriti riga per riga; il ciclo sulle righe è quindi più esterno a quello sulle colonne.

```
For I=1 To M Step 1
   For J=1 To N Step 1
      Mat(I,J) = InputBox("Inserire l'elemento di riga " & I & " e colonna " & J, "Caricamento matrice")
```

```
    Next J
  Next I
```

- per colonne: gli elementi vengono inseriti colonna per colonna; il ciclo sulle colonne è quindi più esterno a quello sulle righe.

```
For J=1 To N Step 1
   For I=1 To M Step 1
      Mat(I,J) = InputBox("Inserire l'elemento di riga " & I & " e colonna " & J, "Caricamento matrice")
   Next I
Next J
```

Si deve ovviamente usare Val se i dati sono numerici.

Si può effettuare il caricamento anche attraverso un controllo DataGridView.
Sia dgvMatrice il nome della DataGridView:

```
      For I=1 To M Step 1
         For J=1 To N Step 1
            Mat (I,J) = Val(dgvMatrice.item(I-1,J-1).value)
         Next J
      Next I
```

Non è detto che la matrice debba essere caricata attraverso l'input. Il caricamento può avvenire utilizzando numeri generati in modo pseudo-casuale, ad esempio con la seguente routine:

```
Dim generatore as New Random      'definisce un oggetto che genera numeri pseudo-casuali
For I=1 To M Step 1
 For J=1 To N Step 1
    Mat(I,J) = generatore.Next(A, B)     'genera un numero intero tra A e B-1
 Next J
Next I
```

In generale, si possono generare numeri pseudo-casuali con queste istruzioni:

- **generatore.Next**: restituisce un numero casuale intero non negativo a 32 bit;
- **generatore.Next(N)**: restituisce un numero casuale compreso tra **0** e **N-1**;
- **generatore.Next(m, M+1)**: restituisce un numero random intero compreso tra **m** e **M**;
- **generatore.NextDouble**: restituisce un numero random con la virgola compreso tra **0.0** e **1.0**.

Vi sono casi in cui per il caricamento di una matrice non è conveniente usare due cicli.
Ad esempio, se in una matrice di tipo stringa, di nome Rubrica, devono essere caricati cognome, nome e numero di telefono di N persone, quindi N righe e 3 colonne, la fase di caricamento potrebbe essere scritta in tal modo, vista la necessità di far comparire il giusto messaggio per ogni elemento da inserire:

```
For I=1 To N Step 1
  Rubrica(I,1) = InputBox ("Inserire il cognome della " & I & "^ persona", "Caricamento rubrica")
  Rubrica(I,1) = InputBox ("Inserire il nome della " & I & "^ persona", "Caricamento rubrica")
  Rubrica(I,1) = InputBox ("Inserire il numero di telefono della " & I & "^ persona", "Caricamento rubrica")
Next I
```

Visualizzazione di una matrice

Per visualizzare tutti i dati contenuti in una matrice, è ancora necessario utilizzare due cicli. La visualizzazione può essere effettuata, ad esempio, all'interno di un controllo **DataGridView**. Anche la visualizzazione può avvenire per riga o per colonna, innestando opportunamente i cicli uno dentro l'altro.

Se dgvMatrice è il nome dato al controllo DataGridView, e il nome della matrice è Mat, con M righe e N colonne, il codice potrebbe essere il seguente:

```
Dim r, c As Byte
'Imposta il numero delle colonne e il numero delle righe
dgvMatrice.RowCount = M
```

```
dgvMatrice.ColumnCount = N
dgvMatrice.RowHeadersWidth = 50
'Riempie la griglia con i dati della matrice
For r = 1 To M
   For c = 1 To N
       dgvMatrice.Item(c - 1, r - 1).Value = Mat(r, c)
   Next
Next
'Riempie la colonna delle intestazioni di riga
For r = 1 To M
   dgvMatrice.Rows(r - 1).HeaderCell.Value = CStr(r)
Next
'Riempie la riga delle intestazioni di colonna
For c = 1 To N
   dgvMatrice.Columns(c - 1).HeaderCell.Value = CStr(c)
Next
`Imposta la larghezza delle colonne
For c = 1 To N
   dgvMatrice.Columns(c - 1).Width = 50
Next
```

	1	2	3
▶ 1	20	3	4
2	51	65	2
✳ 3	3	30	12

I record

Un record è un dato strutturato composto da dati non tutti dello stesso tipo.
Prima di definire una variabile di tipo record, occorre definire la struttura del record, dandole un nome e specificando nome e tipo di ogni elemento (denominato **campo**) che lo compone.
La sintassi della dichiarazione di tipo record è la seguente

```
Structure NomeTipoRecord
   Dim NomeCampo1 As TipoCampo1
   Dim NomeCampo2 As TipoCampo2
   ...
End Structure
```

Es.:

```
Structure RecPersona
      Dim Matricola As String
      Dim Cognome As String
      Dim Nome As String
      Dim DataNascita As Date
End Structure
```

Ai campi di tipo stringa si può imporre un'occupazione di spazio fissa, anteponendo a Dim
<VBFixedString(n)>
Dove n è la lunghezza massima imposta (n deve essere una costante).

Es.:

```
Structure RecPersona
      <VBFixedString(4)> Dim Matricola As String
      <VBFixedString(20)> Dim Cognome As String
      <VBFixedString(20)> Dim Nome As String
      Dim DataNascita As Date
End Structure
```

Definito il tipo record, che si comporta come qualunque tipo di dati predefinito, è possibile utilizzarlo nelle dichiarazioni delle variabili con la clausola Dim:

Es.:

```
Dim Persona As RecPersona
```

La variabile definita di tipo record si compone così di più campi di vario tipo. Per far riferimento ad un campo specifico, si scrive il nome della variabile seguito da un punto e dal nome del campo.

Es.:

```
Persona.Cognome
Fa riferimento al campo Cognome della variabile Record Persona.
```

Dichiarando un vettore di tipo record, si potrà implementare una tabella di dati in modo più semplice rispetto all'uso di più vettori paralleli (è sufficiente un solo vettore).

I campi di un tipo record possono essere anche di tipo strutturato, ad esempio di un altro tipo record o di tipo array.

I Sottoprogrammi

Un sottoprogramma è un gruppo di istruzioni individuate da un nome, tramite il quale se ne può invocare l'esecuzione in qualsiasi momento.
In Visual Basic 2008 i programmi sono generalmente organizzati come collezioni di sottoprogrammi, che possono essere essenzialmente di due tipi:

- **sottoprogrammi legati al verificarsi di eventi relativi ad oggetti**
  `Es. Private Sub btnEsegui_Click`
- **sottoprogrammi definiti dall'utente**, che vengono eseguiti esclusivamente in seguito ad una "chiamata" che avviene tramite il loro nome.

Di seguito si farà riferimento esclusivamente ai sottoprogrammi del secondo tipo, anche se gran parte dei concetti che saranno esposti sono comuni a tutti i tipi di sottoprogrammi.
Un sottoprogramma può richiamare un qualsiasi altro sottoprogramma, anche se stesso. In questo caso si ottiene una **Ricorsione**.

L'utilizzo dei sottoprogrammi offre considerevoli vantaggi:

- se un blocco di istruzioni va eseguito più volte, in punti diversi del programma, utilizzare un sottoprogramma per contenerle permette di scriverle una sola volta;
- un sottoprogramma può essere facilmente riutilizzato in altri programmi.
- un problema complesso può essere suddiviso in sotto-problemi più semplici, da risolvere attraverso sottoprogrammi.

Quando un sottoprogramma viene chiamato, esso non ha dati in comune con il chiamante, ad esclusione delle eventuali variabili globali. La comunicazione fra chiamante e chiamato può però avvenire anche mediante il **passaggio di parametri**.
I parametri sono variabili che il chiamante, in cui prendono il nome di **parametri attuali**, scambia con il sottoprogramma chiamato, in cui prendono il nome di **parametri formali**.
I parametri attuali sono definiti nel programma chiamante e vengono indicati nell'istruzione di chiamata del sottoprogramma. I parametri formali sono collocati nell'intestazione del sottoprogramma stesso.
Il primo parametro attuale deve essere dello stesso tipo del primo parametro formale; il secondo parametro attuale dello stesso tipo del secondo parametro formale, e così via.
Il numero dei parametri attuali deve coincidere con quello dei corrispondenti parametri formali: al momento della chiamata del sottoprogramma, il primo parametro attuale è messo in relazione con il primo parametro formale, il secondo parametro attuale è messo in relazione con il secondo parametro formale e così via. **Non è necessario che un parametro attuale e il corrispondente parametro formale abbiano lo stesso nome**, anzi è meglio che abbiano nomi diversi.

Esistono sostanzialmente due modi per comunicare attraverso i parametri:

- **passaggio per valore**: il valore del parametro attuale viene copiato nel parametro formale corrispondente; con questa modalità si comunicano alcuni valori al sottoprogramma, che poi li potrà utilizzare, ma senza modificarli.
- **passaggio per indirizzo**, anche detto **per riferimento** o **per variabile**: il parametro formale e il parametro attuale condivideranno la stessa area di memoria; modifiche apportate sul parametro formale si ripercuoteranno direttamente sul parametro attuale.

In VB 2008 si deve anteporre al parametro formale la parola **ByRef** per passarlo per indirizzo e **ByVal** per passarlo per valore. Se non si specifica niente, VB 2008 antepone sempre ByVal.
Vettori e Matrici possono essere passati solo per indirizzo, per evitare la duplicazione degli elementi in un'altra struttura.

È possibile scrivere un sottoprogramma all'interno della form class, allo stesso livello di tutti gli altri sottoprogrammi (non all'interno di uno di essi). Ogni sottoprogramma è racchiuso tra un'intestazione di apertura e un'istruzione di chiusura:

```
Public/Private Sub NomeSottoprogramma  (ByVal/ByRef pf1 As tipo1,
                                        ByVal/ ByRef pf2 As tipo2, … )
    …
    …
End Sub
```

Con pf1, pf2, ... sono stati indicati i parametri formali, ciascuno dei quali è preceduto da ByRef o ByVal e seguito dalla dichiarazione del tipo.

Per chiamare un sottoprogramma si scrive

Call NomeSottoprogramma **(**pa1**,** pa2**, ...)**

oppure

NomeSottoprogramma pa1**,** pa2**, ...**

Con pa1, pa2,... sono stati indicati i parametri attuali.

```
Esempio: Il sottoprogramma

    Private Sub Dividi(ByVal A As Int16, ByVal B As integer, ByRef Q AsInt16,
                                                                    R As Int16)
        Q = A\B
        R = A mod B
    End Sub

calcola il quoziente e il resto della divisione tra due numeri interi A e B. La
chiamata può avvenire nel seguente modo:
                Call Dividi(Dividendo, Divisore, Quoziente, Resto)

dove Dividendo, Divisore, Quoziente, Resto sono quattro variabili che devono
essere così dichiarate nel chiamante:

                  Dim Dividendo, Divisore, Quoziente, Resto As Int16
```

I parametri attuali possono anche essere costanti.

```
Esempio: Il sottoprogramma dell'esempio precedente può essere così chiamato
                      Call Dividi(143, 8, Quoziente, Resto)
```

Per i vettori non si specifica l'estremo superiore dell'indice, ma si scrive solo il nome del vettore, nella chiamata, e il nome seguito da () e dalla specifica del tipo, nell'intestazione del sottoprogramma.

```
Il sottoprogramma

       Private Sub CalcolaSomma(ByRef V() As Single, ByVal N As integer,
                                                        ByRef Somma As Single)
            Dim  I As integer
            Somma = 0
            For I = 1 to N
                   Somma = Somma + V(I)
            Next
       End Sub

calcola la somma degli elementi di un vettore numerico e può essere chiamato
tramite l'istruzione
                        Call CalcolaSomma (VNumeri, N, S)
```

Le funzioni

I sottoprogrammi sin qui descritti vengono normalmente indicati con il nome di **Procedure**, per distinguerli da particolari tipi di sottoprogrammi che prendono il nome di **Funzioni** (**Function**).
La funzione è un sottoprogramma che restituisce un solo risultato. Tale risultato viene restituito nel nome della funzione, che rappresenta a tutti gli effetti una variabile.

Una funzione non viene attivata attraverso un'istruzione riservata di chiamata, ma direttamente nell'ambito di un'istruzione di calcolo, normalmente un'istruzione di assegnazione, o di

un'istruzione di output, dove viene invocata tramite il suo nome. È quindi piena l'analogia con il concetto matematico di funzione.
È necessario che ogni sottoprogramma di tipo funzione contenga almeno una funzione di assegnazione in cui si attribuisce un valore al nome della funzione o un'istruzione **Return valore**, che assegna al nome della funzione il risultato.

L'intestazione di una funzione è la seguente:

Public/Private Function NomeFunzione **(ByVal/ByRef** pf1 **as tipo1,**
ByVal/ ByRef pf2 **as tipo2, …) As TipoFunzione**
…
NomeFunzione = … oppure **Return …**
…
End Function

Si osservi che anche al nome della funzione, essendo a tutti gli effetti una variabile, è attribuito un tipo.

Istruzioni di chiamata saranno invece, ad esempio:

v = **NomeFunzione (**pa1, pa2, ... **)**

dove v è il nome di una variabile a cui si assegna il risultato della funzione, oppure

lblRisultato.Text = NomeFunzione (pa1, pa2, ...**)**

dove lblRisultato è il nome di una label a cui si assegna il risultato della funzione, per l'output.

Esempio: La funzione

```
Private Function Somma(V() As Single, N As integer) As Single
     Dim  I As integer
     Somma = 0
     For I = 1 to N
            Somma = Somma + V(I)
     Next
End Function
```

calcola la somma dei valori contenuti in un vettore. La chiamata potrà essere effettuata nel seguente modo:

```
S = Somma(VNumeri(), N)
```

dove VNumeri e N devono comparire nel chiamante in istruzioni di dichiarazione come le seguenti:

```
Dim N As Byte
ReDim VNumeri(1 T N) As Single
```

La funzione

```
Private Function Fattoriale(N As Byte) As Int32
  Dim  I As Byte, F As Long
  F=1
  For I=2 to N
       F=F*I
  Next
  Return F
End Function
```

calcola il fattoriale di un numerodi tipo Byte. La chiamata può essere effettuata anche in un'istruzione di output del tipo:
lblFattoriale=Fattoriale(N), dove N deve essere così dichiarata nel chiamante:

```
Dim N As Byte
```

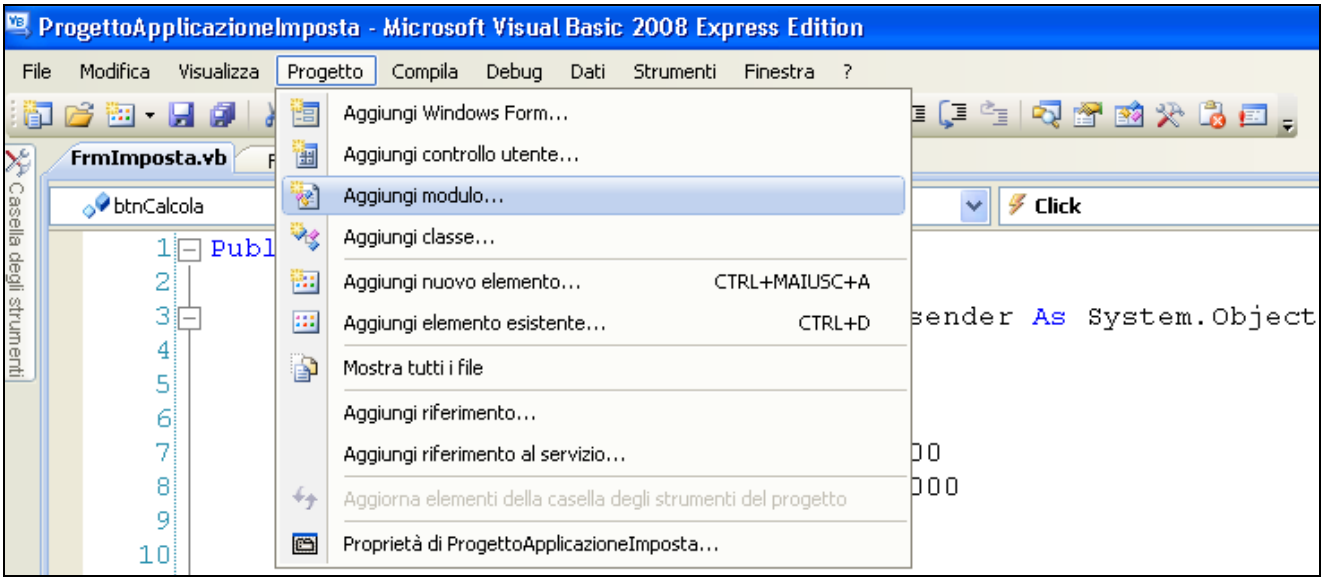
Si osservi che in questo caso si è usato N sia come parametro attuale che come parametro formale (scelta non obbligatoria).

Istruzioni Exit Function e Exit Sub

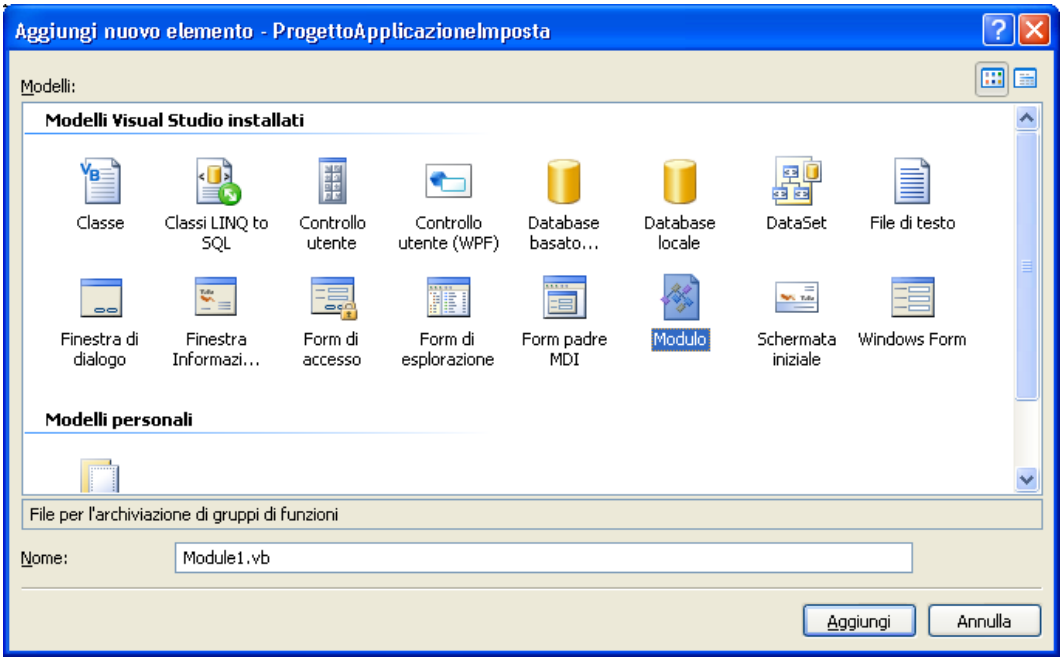
Consentono di uscire prematuramente da una routine e di trasferire immediatamente il controllo all'istruzione successiva alla chiamata della routine. L'esecuzione continua con l'istruzione successiva all'istruzione che ha chiamato la routine.

Visibilità di variabili e sottoprogrammi

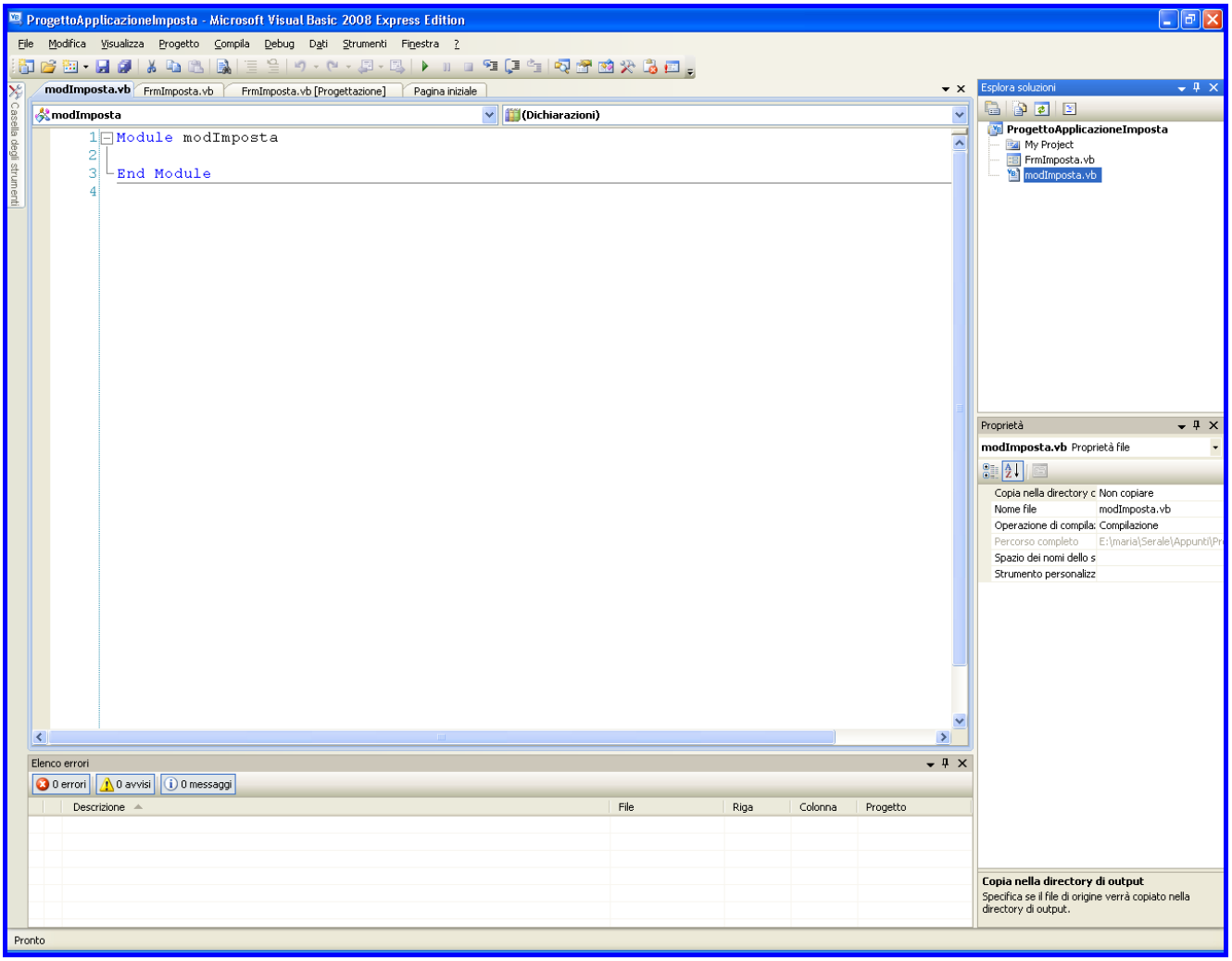
I sottoprogrammi possono essere dotati di un insieme di variabili, che hanno validità solo locale; esse sono definite all'interno del sottoprogramma mediante la clausola **Dim** e prendono il nome di **variabili locali**.
Le variabili definite con **Dim** in un Modulo o nell'area generale di un Form sono **variabili globali** a tutti i sottoprogrammi posti nel modulo o nel form.
Per definire variabili globali per tutto un progetto occorre invece dichiararle con la clausola **Public**, come si fa con i sottoprogrammi.
Un sottoprogramma può essere definito all'interno di un form o all'interno di un modulo. Un **modulo** è un file con estensione **.vb** usato per dichiarare variabili e sottoprogrammi, generalmente per renderli disponibili a più form di uno stesso progetto o a più progetti (dichiararli con la clausola **Public**). Un modulo contenente vari sottoprogrammi pubblici costituisce una **libreria**. Per creare un modulo, selezionare, dalla barra dei menu, **Progetto – Aggiungi modulo...**



Compare la finestra



Selezionare Modulo e modificare eventualmente il nome (viene proposto Module1.vb; bene usare anche in questo caso un prefisso in minuscolo, come **mod**, seguito da una parola significativa per il problema che si sta trattando, con l'iniziale in maiuscolo. Cliccando su **Aggiungi**, compare la finestra del codice riservata al modulo e il nome del file si aggiunge alla finestra **Esplora soluzioni**:



All'interno di un modulo possono essere dichiarate variabili globali per l'intero progetto (con **Public**, anziché **Dim**).
Un sottoprogramma, a sua volta, può essere pubblico (**Public**) o privato (**Private**).
Un sottoprogramma privato può essere richiamato solo nell'ambito del Form o del Modulo in cui è stato definito; un sottoprogramma pubblico può essere richiamato anche da tutti gli altri Moduli e/o Form del progetto.

Funzioni predefinite: librerie del Visual Basic

Il Visual Basic, come altri linguaggi, mette a disposizione una serie di librerie di sottoprogrammi di utilità, prevalentemente di tipo funzione.

Funzioni matematiche

La libreria Math include tutta una serie di funzioni matematiche

Abs(x)	Restituisce il valore assoluto del numero x specificato.
Cos(x)	Restituisce un valore di tipo Double che contiene il coseno dell'angolo x specificato, anch'esso di tipo Double.
Exp(x)	Restituisce un valore Double che specifica *e* (base dei logaritmi naturali) elevato all'esponente x specificato, anch'esso di tipo Double.
Log(x)	Restituisce un valore di tipo Double che contiene il logaritmo naturale (in base *e*) del numero x specificato, anch'esso di tipo Double.
Log10(x)	Restituisce un valore di tipo Double che contiene il logaritmo decimale (in base 10) del numero x specificato, anch'esso di tipo Double.
Int(x)	Restituisce la parte intera del numero x specificato.
Round(x,n)	Restituisce un valore di tipo Double che contiene il numero più prossimo al valore x specificato, anch'esso di tipo Double, arrotondato ad n (di tipo long) cifre decimali.
Sin(x)	Restituisce un valore di tipo Double che specifica il seno dell'angolo x specificato, anch'esso di tipo Double.
Sqr(x)	Restituisce un valore di tipo Double che specifica la radice quadrata di un numero, anch'esso di tipo Double.
Tan(x)	Restituisce un valore di tipo Double che specifica la tangente di un angolo, anch'esso di tipo Double.

```
Es.:  Y = Math.Round(Math.Cos(X), 3))
      Calcola il coseno di X e arrotonda il risultato alla terza cifra decimale.
      Assegna il risultato alla variabile Y.

Es.:  RQ = Math.Round(N, 2)
      Restituisce il valore di N arrotondato a due cifre decimali.
```

Funzioni per la gestione delle stringhe

Per compatibilità verso il basso, VB 2008 continua ad utilizzare le funzioni di VB6; alcune devono essere precedute da **Microsoft.VisualBasic.**, per evitare che i nomi delle funzioni si confondano con altre parole chiave del nuovo linguaggio.
I caratteri di una stringa sono numerati a partire da 1.

Len(s)	Restituisce la lunghezza della stringa s, cioè il numero di caratteri che la compongono.
Microsoft.VisualBasic.Left(s,n)	Restituisce i primi n caratteri a sinistra della stringa s (letti da sinistra a destra).
Microsoft.VisualBasic.Right(s,n)	Restituisce i primi n caratteri a destra della stringa s (letti da sinistra a destra).
Mid(s,n,m)	Restituisce una sottostringa della stringa s composta dagli m caratteri a partire dalla posizione n.

Val(s)	Restituisce la stringa s convertita in formato numerico, se contiene solo numeri. Se contiene anche caratteri, preleva solo le eventuali cifre numeriche all'inizio, altrimenti restituisce 0.
Str(n)	Restituisce il numero n convertito in formato stringa.
Asc(s)	Restituisce il codice Ascii del carattere (o del primo carattere) contenuto nella stringa s.
Chr(n)	Restituisce il carattere il cui codice Ascii è n.
LTrim(s) **RTrim(s)** **Trim(s)**	Restituisce la stringa s privata di eventuali spazi sulla sinistra (LTrim), sulla destra (RTrim) o da ambo le parti (Trim)
Replace(s,s1,s2)	Sostituisce nella stringa s la stringa s2 al posto della stringa s1. È una delle poche procedure.
Instr(n,s1,s2)	Restituisce un valore Integer che specifica la posizione di inizio della prima occorrenza, a partire dalla posizione n, della stringa s2 nelle stringa s1; il valore è 0 se la stringa non è presente.
IsNumeric	Restituisce un valore Boolean in cui è specificato se il valore di un'espressione può corrispondere a un numero (*True* se è un numero, *False* se non lo è).
Ucase(s)	Restituisce s con tutti i caratteri in maiuscolo.
Lcase(s)	Restituisce s con tutti i caratteri in minuscolo.
Space(n)	Restituisce una stringa formata da n spazi bianchi.

```
Es.: Trim("     Luca    ")  restituisce la stringa "Luca"
Es.: Replace("Pippo è un personaggio Disney", "Pippo", "Pluto")
     restituisce la stringa "Pluto è un personaggio Disney".
Es.: Instr(1,"Pippo è un personaggio Disney", "un") restituisce 9
```

Con VB.NET le stringhe sono diventate vere e proprie classi che sono stati dotate di importanti metodi che si vanno ad aggiungere alle funzioni classiche di VB6. In tali metdi, i caratteri di una stringa sono numerati a partire da 0.

S.Chars(N)	Restituisce il carattere di S in posizione N.
S1.Contains(S2)	Restituisce True se la stringa S1 contiene la stringa S2, che può essere anche un carattere.
S1.EndsWith(S2)	Restituisce True se la stringa S1 termina con la stringa specificata (S2).
String.Equals(S1, S2)	Restituisce True se S1 e S2 sono uguali.
S1.IndexOf(S2)	Restituisce l'indice della prima occorrenza di S2 in S1
S1.IndexOf(S2,i)	Restituisce l'indice della prima occorrenza di S2 in S1 a partire dalla posizione i
S1.IndexOf(S2,i,n)	Restituisce l'indice della prima occorrenza di S2 in S1 a partire dalla posizione i e per un numero n di caratteri.
String.IsNullOrEmpty(S)	Restituisce True se la stringa è NULL o vuota.
S.ToUpper	Cambia le lettere di S in maiuscole; equivale alla funzione UCase di VB6.
S.ToLower	Cambia le lettere di S in minuscolo; equivale alla funzione LCase di VB6.

S.Length	Determina il numero di caratteri di una stringa; equivale alla funzione Len di VB6.
S.SubString(i,n)	Restituisce n caratteri della stringa S, a partire dalla posizione i.
S.Remove(i,n)	Rimuove n caratteri della stringa S, a partire dalla posizione i.
S1.Insert(i,S2)	Inserisce la stringa S2 all'interno della stringa S1, a partire dalla posizione i.

```
Es.:  S="Verticale"
      C=S.Chars(3)   assegna alla variabile C il quarto carattere di S, cioè "t"
Es.:  Dim Cognome as String, L as Byte
      Cognome="Rossi"
      L=Cognome.Length()  assegna alla variabile L il valore
Es.:  Dim Cognome as String, CognomeNuovo as String
      Cognome="Rossi"
      CognomeNuovo=Cognome.Insert(0,"De ")  assegna alla variabile Cognome Nuovo
                                            la stringa "De Rossi"
```

Funzioni per la conversione di tipo

Spesso è necessario eseguire la conversione di un dato da un tipo ad un altro (casting). È necessario effettuare la conversione di tipo, ad esempio, quando si vuole applicare una funzione che opera su un determinato tipo di dato.
Il Framework .NET è in grado di eseguire **conversioni automatiche** a runtime verso tipi di ampiezza maggiore, per esempio è in grado di convertire automaticamente Int16 in Int32, Char in String, Single in Double, e via dicendo. Queste operazioni di conversione vengono dette **widening** (wide = largo) e non comportano perdita di dati, poiché trasportano un valore che contiene una data informazione in un tipo che può contenere più informazioni. Per conversioni che vanno nella direzione opposta, dette **narrowing** (narrow = stretto), che possono comportare la perdita di dati e perciò generare errori, occorre far ricordo a specifici operatori di conversione. Le conversioni esplicite sono utili perché consentono di ottenere programmi più efficienti.

CType è una funzione di conversione universale e permette la conversione di qualsiasi tipo in qualsiasi altro tipo, almeno quando questa è possibile. La sintassi è molto:

Variabile = CType(Valore, Tipo)

Ad esempio:

```
Dim I As Int32, B As Byte
B = CType(I, Byte)
```

È bene usare esplicitamente l'operatore di conversione CType per convertire

- un valore intero o decimale in uno booleano;
- un valore Single o Double in Decimal;
- un valore intero con segno in uno senza segno;
- un valore intero senza segno in uno con segno della stessa ampiezza (ad esempio da UInt32 a Int32).

Oltre a CType, esistono operatori di cnversione in un tipo specifico: **CInt** converte in Int32, **CBool** in Booleano, **CByte** in Byte, **CShort** in Int16, e così via con gli altri, **CLong**, **CUShort**, **CULong**, **CUInt**, **CSng**, **CDbl**, **CDec**, **CStr** (o anche **Str**), **CDate**, **CObj**.

Ad esempio:

```
Dim I As Int32, B As Byte
B = CByte(I)
```

Per le conversioni in stringhe e date, VB 2008 mette a disposizione classi con proprietà e metodi che consentono funzionalità molto avanzate. Per le stringhe, poi, è disponibile, come metodo, la funzione ToString per variabili di qualunque tipo.

Ad esempio:

```
Dim B As Byte, S as string
S = B.ToString
```

La gestione delle date

Una variabile di tipo Date è un oggetto che rappresenta un istante di tempo, tipicamente espresso come data e ora di un giorno. Per la gestione del suo contenuto sono disponibili funzioni e metodi. Sia d una variabile di tipo Date:

d.Date	Restituisce la sola data contenuta in d.
d.Day	Restituisce il giorno della data contenuta in d.
d.DayOfWeek	Restituisce il giorno della settimana del giorno della data contenuta n d (1÷7).
d.DayOfYear	Restituisce il giorno dell'anno corrispondente alla data contenuta n d (1÷366).
d.Month	Restituisce il mese della data contenuta in d.
d.Year	Restituisce l'anno della data contenuta in d.
d.TimeOfDay	Restituisce l'ora completa (ore, minuti, secondi, millesimi di secondo) contenuta in d; è un dato di tipo TipeSpan.
d.Hour	Restituisce solo il numero delle ore contenuto in d.
d.Minute	Restituisce solo il numero dei minuti contenuto in d.
d.Second	Restituisce solo il numero dei secondi contenuto in d.
d.Add	Restituisce il valore di d aumentato di un valore di tempo specificato (TipeSpan).
d.AddDays	Restituisce il valore di d aumentato di un numero di giorni specificato.
d.AddHours	Restituisce il valore di d aumentato di un numero di ore specificato.
d.AddMinutes	Restituisce il valore di d aumentato di un numero di minuti specificato.
d.AddMonths	Restituisce il valore di d aumentato di un numero di mesi specificato.
d.AddSeconds	Restituisce il valore di d aumentato di un numero di secondi specificato.
d.AddYears	Restituisce il valore di d aumentato di un numero di anni specificato.

Infine la funzione **Now** restituisce data e ora correnti.

Le costanti di tipo Date devono essere specificate racchiuse tra #

Esempi:

```
        Dim d As Date, dd As Date, t As TimeSpan
        Dim g, m, hh, mm, ss, aa, gs, ga As Byte, a, ms As Int16
        d = Now  'in d si assegnano data e ora attuali
        dd = d.Date 'in dd si assegna solo la data odierna
        g = d.Day 'in g si assegna solo il giorno della data attuale
        m = d.Month 'in m si assegna solo il mese della data attuale
        a = d.Year 'in a si assegna solo l'anno della data attuale
        t = d.TimeOfDay 'in t si assegna solo il tempo
        hh = d.Hour 'in hh si assegna solo il numero delle ore
        mm = d.Minute 'in mm si assegna solo il numero dei minuti
        ss = d.Second 'in ss si assegna solo il numero dei secondi
        lbl20Giorni.Text = d.AddDays(20) 'visualizza la data tra 20 giorni
```

I File

Visual Basic 2008 permette di creare e gestire files sia ad accesso sequenziale che ad accesso casuale.

I file ad accesso sequenziale

Un file, di qualsiasi tipo esso sia, per poter essere utilizzato deve necessariamente essere aperto. L'apertura si esegue con l'istruzione **FileOpen**, la cui sintassi, per le strutture ad accesso sequenziale, è la seguente:

FileOpen (*numeroFile*, *percorso\file*, *modalità*)

dove:

numeroFile è un numero che identificherà il file univocamente all'interno del programma. Tale numero corrisponde al canale (buffer) attraverso cui transiteranno le informazioni tra la memoria centrale e il file e sarà utilizzato come nome interno in tutte le successive operazioni di lettura, scrittura e chiusura del file. Deve essere un intero compreso fra 1 e 511.

percorso\file è una stringa (esplicita, costante o variabile) che indica il percorso (facoltativo) e il nome (obbligatorio) del file che si desidera aprire. E' detto anche nome esterno e va specificato solo al momento dell'apertura.

modalità indica l'intento con cui si accede al file, che non può mutare fino alla chiusura del file stesso. Le possibili modalità sono:

- OpenMode.Output crea un nuovo file vuoto su disco, predisponendolo per la scrittura delle informazioni. Se il file è già presente sul disco, viene prima cancellato.
- OpenMode.Append apre un file esistente su disco, al fine di accodare altri record a quelli già presenti. Se il file non viene trovato, ne viene creato uno nuovo senza alcun messaggio d'errore e il funzionamento coincide in pratica col caso precedente.
- OpenMode.Input il file è aperto in lettura, ovvero è utilizzato esclusivamente per reperire informazioni senza poter effettuare su di esse alcuna modifica. È ovvio che l'apertura di un file in questa modalità richiede che esso sia stato in precedenza creato sul disco, altrimenti provoca un errore che il File System notifica al programma in esecuzione.

Esempi:

```
FileOpen (1, "C:\Personale\Dipendenti.seq", OpenMode.Output)
FileOpen (2, "\Personale\Clienti.seq", OpenMode.Append)
FileOpen (3, ".\Prodotti.seq", OpenMode.Input)
FileOpen (4, "Magazzino.seq", OpenMode.Append)

Const FileO = "C:\Personale\Dipendenti.seq"
FileOpen (1, FileO, OpenMode.Output)

Const FileI = ".\Prodotti.seq"
FileOpen (3, FileI, OpenMode.Input)
```

Per **scrivere dati** in un file sequenziale, si apre il file in modalità OpenMode.Output o OpenMode.Append, si usa l'istruzione:

PrintLine(*numeroFile, dato*)

dove ***numeroFile*** è il numero che identifica il file e ***dato*** è il dato da scrivere nel file.

Esempio:

```
PrintLine (1, Cognome)
PrintLine (1, Nome)
```

Ogni istruzione PrintLine esegue una scrittura di un dato. I dati sono scritti in sequenza e possono essere anche di tipi diversi.

Per **leggere dati** da un file sequenziale, si apre il file in modalità OpenMode.Input e si usa l'istruzione:

dato **= LineInput(*numeroFile*)**

dove ***numeroFile*** è il numero che identifica il file e ***dato*** è il dato da leggere dal file. I dati vengono letti nell'ordine esatto con cui sono stati scritti.

```
Esempio:        Cognome= LineInput (1)
                Nome = LineInput (1)
```

Quando un file viene aperto in lettura, il puntatore di lettura è posizionato all'inizio del primo dato registrato. Ogni lettura di un dato fa avanzare il puntatore all'inizio del dato successivo. **Per sapere se la lettura ha raggiunto la fine del file**, è necessario ricorrere alla funzione **EOF** (**End Of File**), alla quale, come di consueto, va passato il numero del file come parametro. Il valore restituito è di tipo logico e vale *True* se è stata raggiunta la fine del file, *False* in caso contrario.
Si noti che nei file sequenziali EOF diventa *True* non appena viene letto l'ultimo dato.

Per scorrere tutto il contenuto di un file sequenziale, è allora necessario utilizzare un ciclo del tipo:

```
Do While Not EOF(n)
    Dato=LineInput (n)
    <trattamento del dato letto>
Loop
```

Prima di aprire un fie in lettura, il programma deve **accertarsi che il file esista** sul disco. La funzione **Dir(percorso\file)** restituisce una stringa vuota se non trova il file.

Esempio:

```
Private Sub ...
  Const FileI = "Prodotti.seq"     'File di Input
  ...
  If Dir(FileI) = "" Then
     Msgbox "File inesistente: " & FileI, vbCritical, "Attenzione!"
  Else
        FileOpen (1, FileI, OpenMode.Input)
        <lettura e trattamento del file>
  End If
End Sub
```

Quando le operazioni sul file sono terminate, il file deve essere chiuso. Questa operazione svuota il buffer, portando a termine le eventuali scritture in sospeso. Inoltre rilascia il numero di buffer, che diventa riutilizzabile per la gestione di un altro file (*pratica sconsigliata per ragioni di chiarezza: meglio attribuire a ogni nome esterno sempre lo stesso numero interno*). Ciò comunque non pregiudica la successiva riapertura dello stesso file nel medesimo programma, con la stessa o una differente modalità.

La sintassi è:	**FileClose(*numeroFile*)**	per chiudere il file di cui è indicato il numero
oppure semplicemente:	**FileClose()**	per chiudere tutti i file aperti.

```
Esempi:         FileClose(1)
                FileClose(3)
                FileClose()
```

Un file sequenziale non permette che i record vengano sovrascritti per un'eventuale correzione, né che vengano cancellati, né inseriti, se non dopo l'ultimo.
Per raggiungere ugualmente tali obiettivi in un file sequenziale è necessario:

- aprire in lettura il file da modificare;
- aprire in scrittura (output) un file d'appoggio;
- copiare tutti i record dal primo file al secondo, avendo cura di effettuare le correzioni necessarie prima di scrivere nel secondo file;
- chiudere i due file;
- cancellare il primo file con **Kill (*percorso\file*);**
- rinominare il secondo file con il nome che aveva il primo, con **Name (*vecchio_nome* As *nuovo_nome*);**

Un file sequenziale può essere usato per memorizzare anche un testo. In questo caso ogni operazione di lettura/scrittura potrà riguardare, ad esempio, una riga del testo.

I file ad accesso diretto (random)

I file sequenziali presentano un limite: la scarsa idoneità all'utilizzo di grandi quantità di dati, a causa degli elevati tempi di ricerca. I file ad accesso diretto (o casuale, o random) sono invece dotati di una struttura rigida che li rende in grado di offrire maggiore velocità di accesso alle informazioni.

I blocchi d'informazioni (record) registrati in un file random devono avere tutti la stessa struttura e la stessa lunghezza. La struttura di un record è definita, come nuovo tipo di dati, attraverso la frase **Structure**:

```
Public Structure nome_tipo
   Dim nomecampo_1 As tipo_1
   Dim nomecampo_2 As tipo_2
   ...
   Dim nomecampo_n  As tipo_n
End Structure
```

All'interno della struttura devono essere dichiarati gli elementi che la compongono, denominati **campi**. Per ognuno di essi deve essere specificato il nome e il tipo, che può essere di tipo semplice (Byte, Integer, Long, Single, Double, Boolean, stringa a lunghezza fissa) o strutturato (record o array).
Un tipo così definito è denominato **tipo record** e deve essere dichiarato nell'area generale della classe form o in un modulo.
Se un campo è di tipo stringa, deve essere preceduto dalla specifica della sua lunghezza massima, visto che i record devono avere tutti la stessa lunghezza:

<VBFixedString(n)>

Dove n è la lunghezza massima del campo.
Esempio:

```
Structure T_Automobile
   <VBFixedString(15)> Dim Marca As String
   <VBFixedString(30)> Dim Modello As String
   Dim Cilindrata As Integer
   Dim Diesel As Boolean
End Structure
```

In questo modo si è provveduto ad aggiungere un nuovo tipo di dati a quelli standard previsti da Visual Basic. È quindi possibile dichiarare, ad esempio, la variabile *Automobile* di tipo `T_Automobile` digitando:

```
Dim Automobile As T_Automobile
```

Per assegnare valori ad una variabile di tipo record, si esegue un'operazione di assegnazione per ogni campo, secondo la sintassi:

variabile.campo = valore

Ad esempio, volendo assegnare il valore 1700 al campo *Cilindrata* della variabile *Automobile*, definita in precedenza, occorre scrivere:

```
Automobile.Cilindrata = 1700
```

Come per i file sequenziali, l'**apertura di un file ad accesso diretto** avviene per mezzo dell'istruzione **FileOpen** che, in questo caso, assume la forma:

FileOpen (*numeroFile*, *percorso\file*, *modalità*, , , *lunghezzaRecord*)

La modalità di accesso è **OpenMode.Random** qualunque sia l'operazione che si desidera effettuare sul file. Al parametro ***lunghezzaRecord*** deve essere assegnata l'esatta dimensione del record in byte, che può essere determinata per mezzo della funzione **Len**, il cui parametro è una qualunque variabile del tipo record del file. Ad esempio:

```
FileOpen (1, "Concessionaria", OpenMode.Random, , , Len(Automobile))
```

La presenza delle virgole prima del parametro di lunghezza del record è motivato dal fatto che l'istruzione File Open completa prevede altri due parametri facoltativi prima di specificare la lunghezza del record.

Anche i file ad accesso casuale devono essere chiusi dopo l'uso per mezzo dell'istruzione **FileClose**.

Per **scrivere il contenuto di un record** in un file ad accesso casuale si usa l'istruzione:

FilePut (*numeroFile, variabile, posizione*)

dove ***numeroFile*** è il numero del buffer con cui è stato aperto il file, ***variabile*** è il nome della variabile contenente i dati da scrivere e il parametro ***posizione*** indica il numero della posizione in cui scrivere. I record sono numerati a partire da 1.

Esempio:

```
FilePut (1, R, 5)
```

scrive il contenuto della variabile di tipo record *R* nella quinta posizione del file con buffer 1.
Se il valore del parametro *posizione* individua un record già presente in archivio, esso viene sovrascritto. Per aggiungere un nuovo record al file (operazione equivalente all'append dei file sequenziali) è invece necessario indicare il numero del record successivo all'ultimo.
Nei file random è possibile calcolare facilmente il numero dei records presenti nel file (perché hanno tutti la stessa lunghezza) mediante la formula:

numeroDeiRecordsNelFile = lunghezzaDelFileInByte / lunghezzaRecordInByte

La lunghezza del file può essere calcolata usando la funzione **LOF(n)**, dove **n** è il numero del buffer.
La lunghezza di un record può invece essere calcolata attraverso la funzione **LEN(v)**, dove **v** è una variabile del tipo record dichiarato per il file.
Il numero del record successivo all'ultimo si ottiene aggiungendo 1 al numero complessivo dei records:

nr = LOF(n)\LEN(V)+1

La **lettura di un record** contenuto in un file ad accesso casuale avviene per mezzo dell'istruzione:

FileGet (*numeroFile, variabile, posizione*)

dove ***numeroFile*** è il numero del buffer con cui è stato aperto il file, ***variabile*** è il nome della variabile in cui i dati letti devono essere posti e il parametro ***posizione*** indica il numero del record da leggere, compreso fra 1 e il numero dei records presenti nel file. Ad esempio, per leggere il quarto record del file di buffer 1 e porre il suo contenuto nella variabile *R*, l'istruzione è:

```
FileGet (1, R, 4)
```

Solo in un caso il valore del parametro *posizione* può essere omesso: quando le operazioni di lettura avvengono in sequenza; l'assenza del numero indicante la posizione provoca infatti l'accesso al record successivo a quello corrente
Ad esempio, la sequenza:

```
FileGet (1, R, 4)
FileGet (1, V)
```

provoca la lettura, nelle variabili R e V, del quarto e del quinto record del file con buffer 1.

Il metodo più semplice per cancellare un record consiste nel sovrascriverlo con un elemento vuoto. In questo modo, tuttavia, non è possibile recuperare lo spazio da esso occupato sul disco, si tratta cioè di una **cancellazione logica**, non **fisica**. La cancellazione logica dei record offre però un'importante opportunità: fare in modo che elementi eliminati per errore possano essere agevolmente recuperati. Ciò è possibile aggiungendo un campo booleano alla struttura dei record e facendo in modo che il programma che accede all'archivio consideri cancellati tutti gli elementi contenenti il valore logico *True* all'interno di questo campo.
L'eliminazione di un record comporta quindi la semplice variazione del valore di un suo campo. Analogamente, è possibile recuperare un elemento cancellato per errore impostando nuovamente al valore *False* il campo booleano.
La struttura *T_Automobile* può a tal fine essere modificata come segue:

```
Public Structure T_Automobile
   <VBFixedString(20)> Dim Marca As String
   <VBFixedString(50)> Dim Modello As String
   Dim Cilindrata As Integer
   Dim Diesel As Boolean
   Dim Cancellato As Boolean
End Structure
```

Quando la quantità di informazioni da gestire diventa elevata, la necessità di recuperare lo spazio occupato dai record cancellati diventa evidente, sia per liberare spazio su disco, sia per ridurre i tempi di accesso alle informazioni, che aumentano costringendo il programma a leggere dati inutili.
Basta allora realizzare una procedura in grado di creare un nuovo file e copiare al suo interno tutti i record non vuoti. Successivamente, si deve eliminare il vecchio file ed assegnare il suo nome alla nuova struttura.

Scorrere il contenuto di un file random implica la necessità di sapere quando si arriva alla fine del file. Come per i file sequenziali, è disponibile la funzione **EOF**, per cui il ciclo di lettura può, ad esempio, essere il seguente:

```
Do While Not EOF(1)
    FileGet (1, Automobile)
    ' elaborazione dati record letto
    …
Loop
```

Potendo però calcolare il numero dei record nel file, si può anche usare un semplice ciclo For…Next:

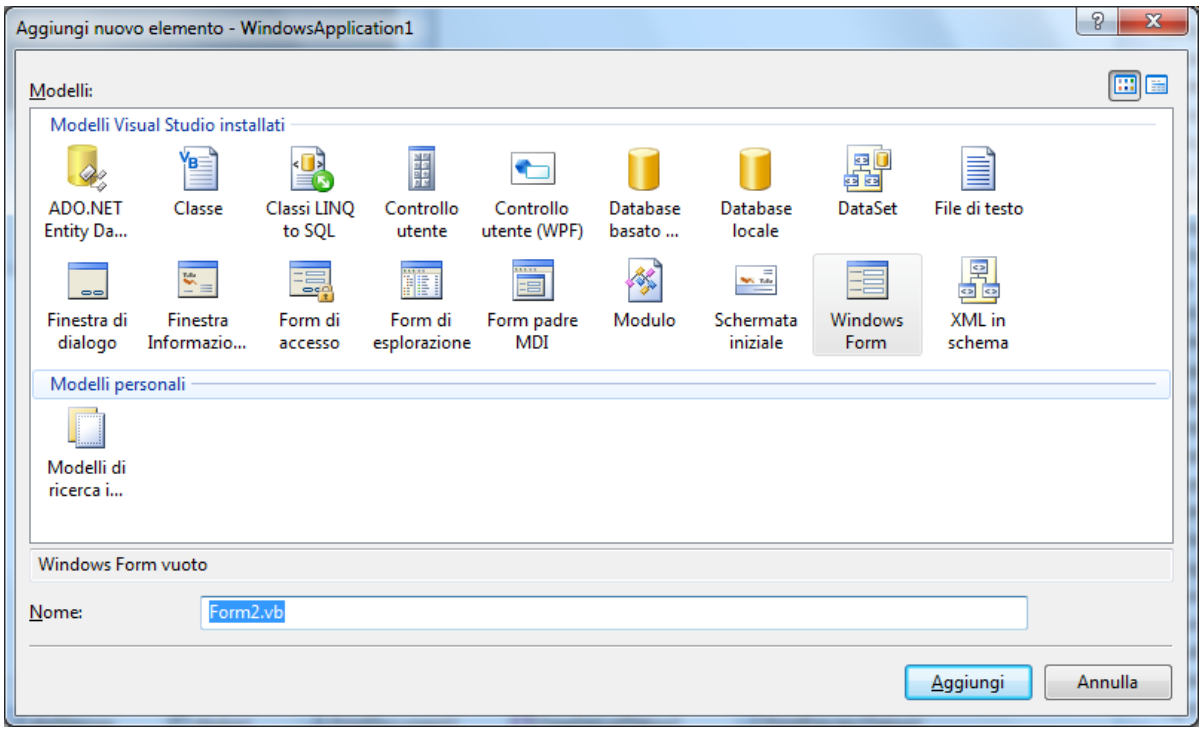
```
NR=LOF(1)/LEN(Automobile)
For I = 1 to NR
    FileGet (1, Automobile, I)
    ' elaborazione dati record letto
    …
Next
```

Progetti multiform

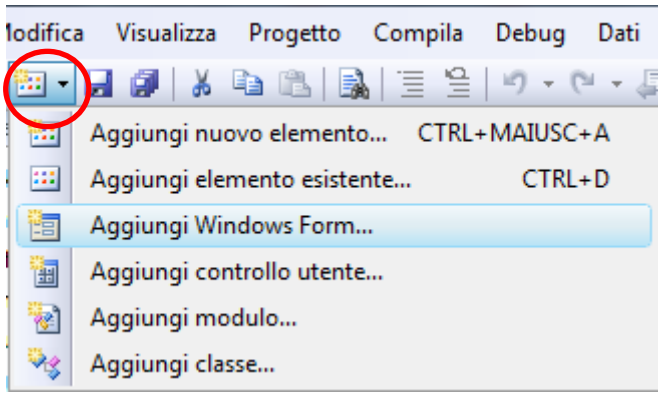
Aggiungere un form ad un progetto

Per aggiungere ad un progetto VB un nuovo form si può procedere in uno dei seguenti modi:

- Selezionare, dal menu di VB, **Progetto – Aggiungi Windows Form…**, selezionare **Windows Form** e cliccare su **Aggiungi** nella finestra che compare.



- Cliccare sulla freccetta a destra del pulsante **Aggiungi Nuovo elemento** sulla barra degli strumenti di VB e selezionare **Aggiungi Windows Form…** dal menu che compare. Se invece si clicca sul pulsante stesso, appare la stessa finestra al punto precedente, da cui si deve selezionare **Windows Form** e cliccare su **Aggiungi**. Il pulsante comunque si trasforma e diventa **Aggiungi Windows Form**.



- fare clic con il tasto destro del mouse sul nome del progetto all'interno del Solution Explorer e scegliere Add (Aggiungi), quindi selezionare **Aggiungi Windows form**.

Allo stesso modo si può procedere per aggiungere un **modulo** al progetto (un modulo è un file con estensione .BAS in cui è possibile dichiarare, con la clausola **Public**, variabili e sottoprogrammi da rendere globali per tutti i form di un progetto multiform).

Sia per i form che per i moduli aggiunti, al successivo salvataggio del progetto sarà richiesto di salvare il file corrispondente.
Fra tutti i form, uno deve essere definito come form di avvio (o principale).
È il form di avvio, per default, il primo form creato. Per cambiare questa impostazione, è possibile selezionare **Progetto** – **Proprietà di *NomeProgetto*** e impostare il nome del nuovo form di avvio in **Form di avvio**.

Metodi ShowDialog e Show

Il metodo ShowDialog carica in memoria centrale un form (se non è già caricato) e lo visualizza. La sintassi è:

NomeForm.ShowDialog()

Per un form visualizzato sullo schermo, la proprietà Visible è impostata su True.
All'avvio di un'applicazione, il form principale verrà caricato e visualizzato automaticamente.
La visualizzazione del form è modale, cioè il Form è visualizzato e l'esecuzione è bloccata finché il form non viene chiuso. Invece, se vi vuole una visualizzazione non modale, è necessario richiamare il metodo Show:

NomeForm.Show()

Metodo Hide

Nasconde un Form, senza scaricarlo dalla memoria centrale. La sintassi è:

NomeForm.Hide

La sua proprietà **Visible** è impostata su **False**. I controlli di un form nascosto non sono accessibili per l'utente, ma sono disponibili per l'applicazione Visual Basic in esecuzione.

Istruzione Close

Scarica dalla memoria un form. La sintassi è:

NomeForm.Close

Quando un form viene scaricato, tutte le modifiche apportate in fase di esecuzione ai controlli e al form verranno perdute, per cui è utile scaricare un form quando è necessario reimpostare le sue proprietà sui valori originali. Un form può scaricare se stesso con l'istruzione

Me.Close

Eventi Activated, Deactivate, Load e Unload del form

In seguito all'azione dei metodi e delle istruzioni illustrate, si possono generare (e quindi programmare) i seguenti quattro eventi relativi ai form:

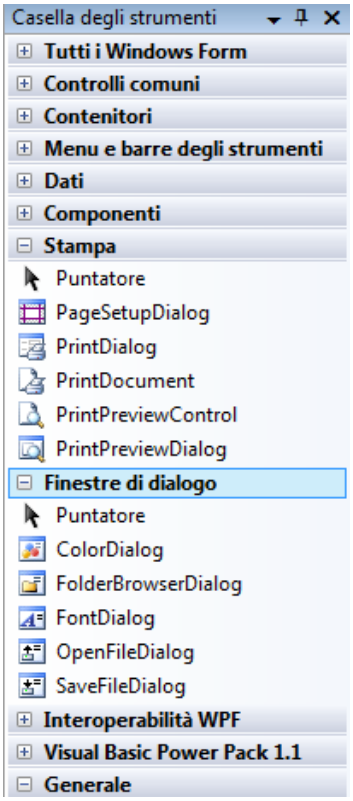
- **Activated**, che viene generato quando un form diventa la finestra attiva, ad esempio attraverso l'esecuzione del metodo Show o del metodo SetFocus o quando l'utente clicca su di esso.
- **Deactivate**, che viene generato quando un form perde lo stato di finestra attiva.
- **Load**, che viene generato quando il form viene caricato, ad esempio mediante l'istruzione Show. Se il form viene nascosto con Hide, esso rimane comunque in memoria centrale. Viene generato prima di Activated, quando il form è caricato in memoria centrale ma non è ancora visibile.
- **Unload**, che viene generato quando il form viene scaricato mediante l'istruzione Close.

L'evento **Activated** viene generato prima dell'evento **GotFocus**. L'evento **LostFocus** viene generato prima dell'evento **Deactivate**.

Le finestre di dialogo (Common Dialog Box)

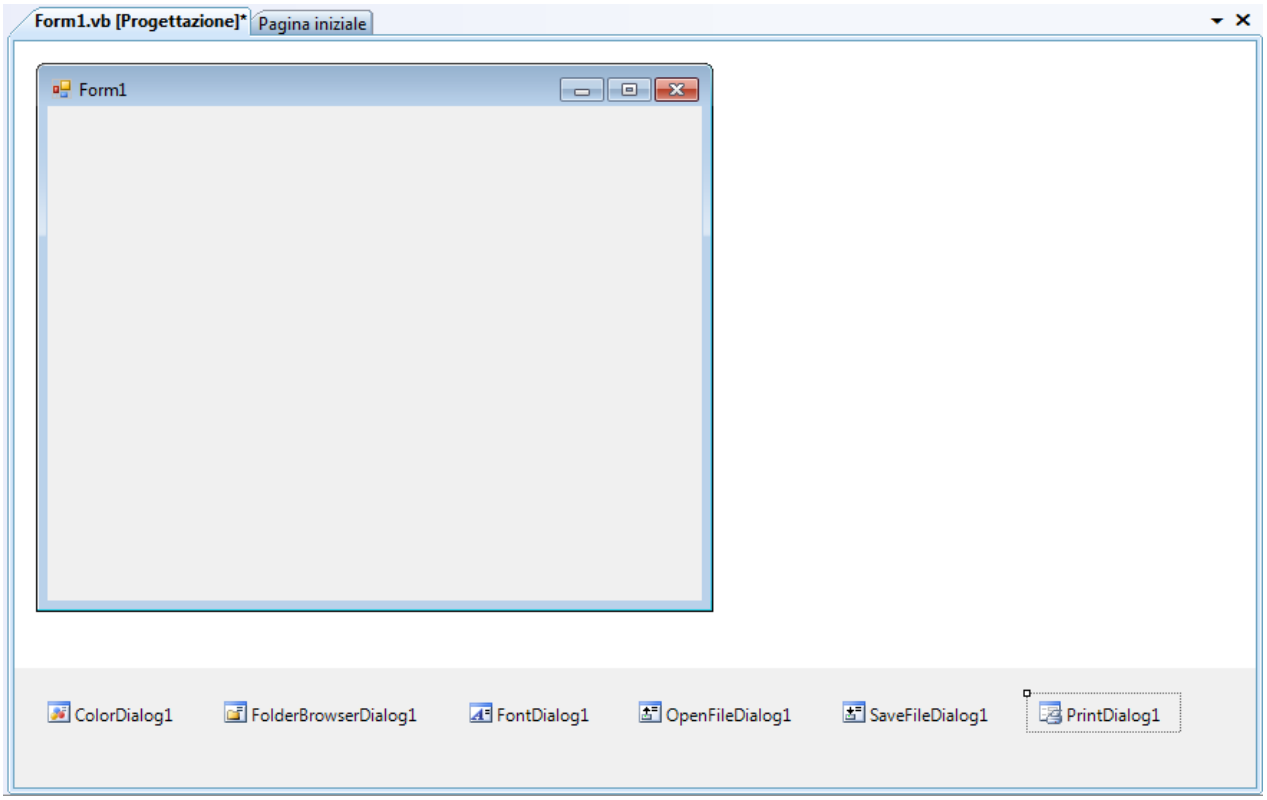
La maggior parte dei programmi che funzionano con Windows presentano la possibilità, per l'utente, di interagire con il programma attraverso alcune finestre dette **Finestre di dialogo comuni** o **Common Dialog Box**. Le finestre di dialogo comuni sono quelle finestre che, come dice il nome stesso, sono standard in tutte le applicazioni per Windows.
Visual Basic mette a disposizione le seguenti finestre di dialogo:

- **apertura file**: è una finestra che consente all'utente di scegliere un file e aprirlo;
- **salvataggio file**: permette all'utente di salvare un file, scegliendone la destinazione;
- **selezione carattere**: dà la possibilità di impostare le caratteristiche dei caratteri;
- **selezione colore**: dà la possibilità all'utente di scegliere un colore;
- **navigazione cartelle**: dà la possibilità all'utente di cercare file e cartelle nel file system (l'utility "sfoglia");
- **selezione stampante**: permette all'utente di scegliere una stampante per stampare un documento e di definire le impostazioni di stampa.



Questo tipo di finestre è detto **modale** perché l'utente deve chiuderle con un click su OK o Annulla prima di continuare il suo lavoro con l'applicazione.

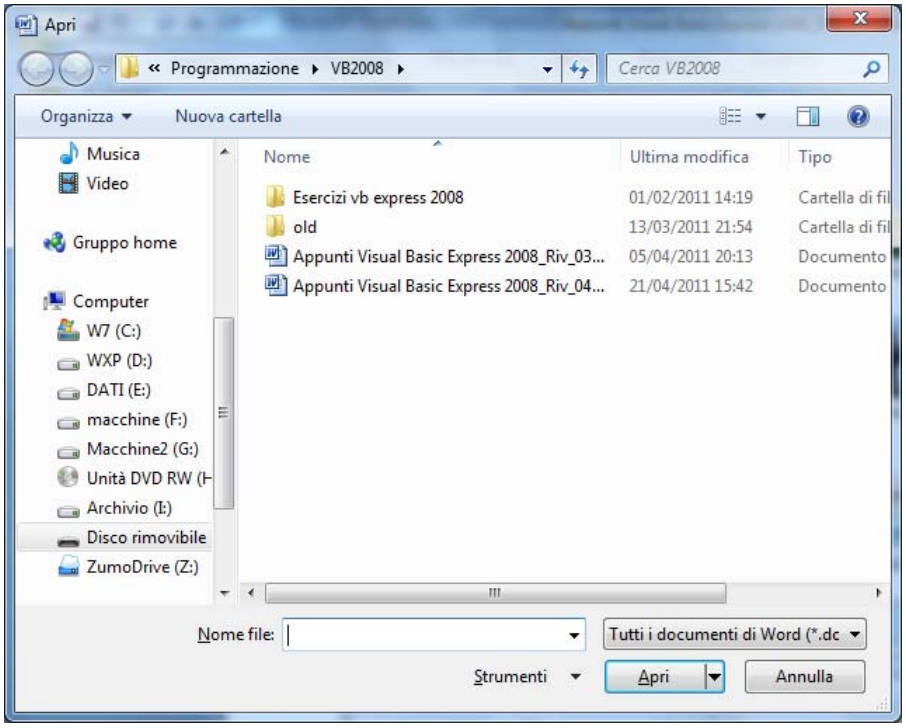
In VB 2008 le finestre di dialogo comuni sono gestite da controlli che fanno parte del corredo di controlli standard; essi sono: **OpenFileDialog, SaveFileDialog, FontDialog, ColorDialog, FolderDialogBox** e **PrintDialog**. I primi cinque controlli fanno parte del gruppo **Finestre di dialogo**, l'ultimo è invece inserito nel gruppo **Stampa**, che presenta tutta una serie di finestre ed oggetti per predisporre la stampa.
Quando uno di questo controlli viene posizionato sul form, non è su questo che compare, ma in un apposito spazio nella parte bassa della finestra di progettazione del form.



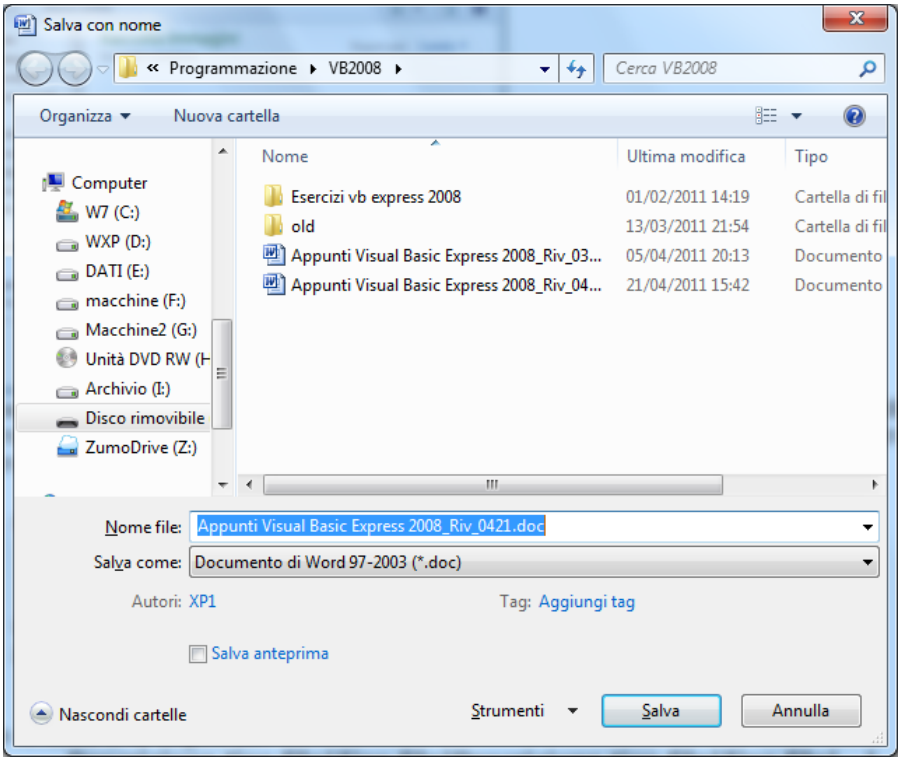
Selezionando un controllo in questo spazio, è possibile impostarne le proprietà attraverso la finestra delle proprietà. Tutti hanno, ad esempio, la proprietà **Name**, in cui attribuire un nome al controllo. Il prefisso dei nomi può essere per tutti **dlg** (dialog).
I controlli non saranno mai visibili sul form, ma compariranno solo se invocati attraverso il metodo **ShowDialog**.

Le finestre di dialogo Apri e Salva

Compaiono rispettivamente attraverso i controlli **OpenFileDialog** e **SaveFileDialog**, che sono molto simili tra loro.



Apri

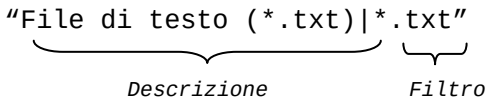


Salva

Per inserire filtri sui tipi di file, si deve utilizzare la proprietà **Filter** della finestra di dialogo.

Un filtro specifica il tipo di file visualizzati nella casella di riepilogo dei file della finestra di dialogo. La proprietà **Filter** è formata da coppie in cui il primo elemento è la descrizione del tipo di file, mentre la seconda è il filtro corrispondente.

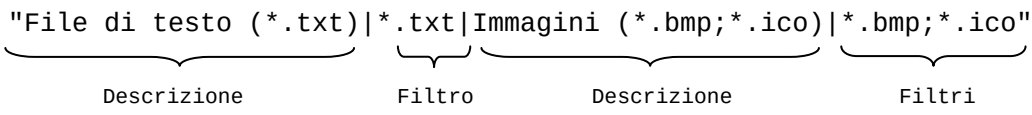
Esempio:




```
Questo filtro fa apparire solo i file con estensione txt.
La stringa "File di testo (*.txt)" appare nella casella che elenca i tipi.
```

E' possibile specificare più filtri corrispondenti ad una stessa descrizione; in questo caso devono essere separati da un punto e virgola, senza spazi:

Esempi:




```
File di testo|*.txt|Tutti i file|*.*

File immagine|*.bmp;*.jpg;*.gif|Tutti i file|*.*
```

Il carattere **| (pipe)** è usato per separare sia le varie coppie, sia la descrizione dal filtro in una stessa coppia. Non si devono includere spazi prima o dopo il simbolo, in quanto tali spazi verrebbero visualizzati insieme ai valori di descrizione e filtro.

Quando si specificano più filtri per una finestra di dialogo, si deve utilizzare la proprietà **FilterIndex** per specificare il filtro predefinito. Il valore 1 specifica il primo, 2 il secondo, e così via.

Le proprietà **CheckFileExists e CheckPathExists**, se impostate su **true**, visualizzano un messaggio di errore se, rispettivamente, il file o il percorso selezionato nella finestra non esiste; tali proprietà sono utili, ad esempio, quando si vuole aprire un file. **InitialDirectory** imposta la cartella che deve essere visualizzata quando si apre la finestra di dialogo.
Il controllo **SaveFileDialog** ha una proprietà molto importante: **OverwritePrompt**, che, se impostata su true, visualizza un messaggio di conferma qualora l'utente specifichi il nome di un file esistente per il salvataggio.

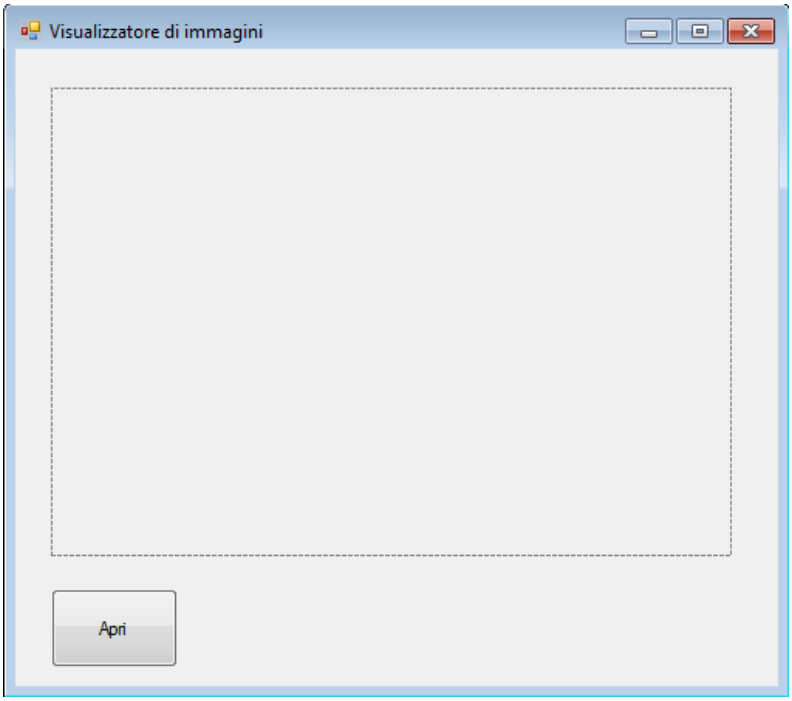
Il metodo **ShowDialog**, che, come detto, visualizza le finestre di dialogo, restituisce, per le finestre Apri e Salva, un valore che può essere utilizzato per sapere se l'utente ha effettivamente selezionato un file oppure ha premuto il pulsante **Annulla**, quindi ha interrotto l'operazione. Se il valore restituito dal metodo **ShowDialog** è uguale a **DialogResult.OK**, significa che l'utente ha selezionato un file e premuto il pulsante **OK**: in questo caso la proprietà **FileName** del controllo contiene il nome del file. Se, invece, l'utente ha premuto **Annulla**, **ShowDialog** varrà **DialogResult.Cancel**.

La proprietà **Title** consente di impostare il titolo della finestra di dialogo.

E' da notare che i controlli **OpenFileDialog** e **SaveFileDialog** si limitano a fornire un'interfaccia comune per selezionare il file da aprire o salvare; le operazioni di apertura e salvataggio vere e proprie dovranno essere poi realizzate dal programmatore. Se si vuole avere piena libertà nell'utilizzo delle finestre nel proprio software, può essere opportuno porre a false le proprietà **CheckFileExists**, **CheckPathExists** e **OverwritePrompt**, al fine di personalizzare la messaggistica da produrre in caso di errore (file inesistente o file da sovrascrivere) e le azioni da realizzare in corrispondenza.

Esempio: programma per selezionare e visualizzare immagini.

Viene usato un controllo OpenFileDialog di nome dlgApri

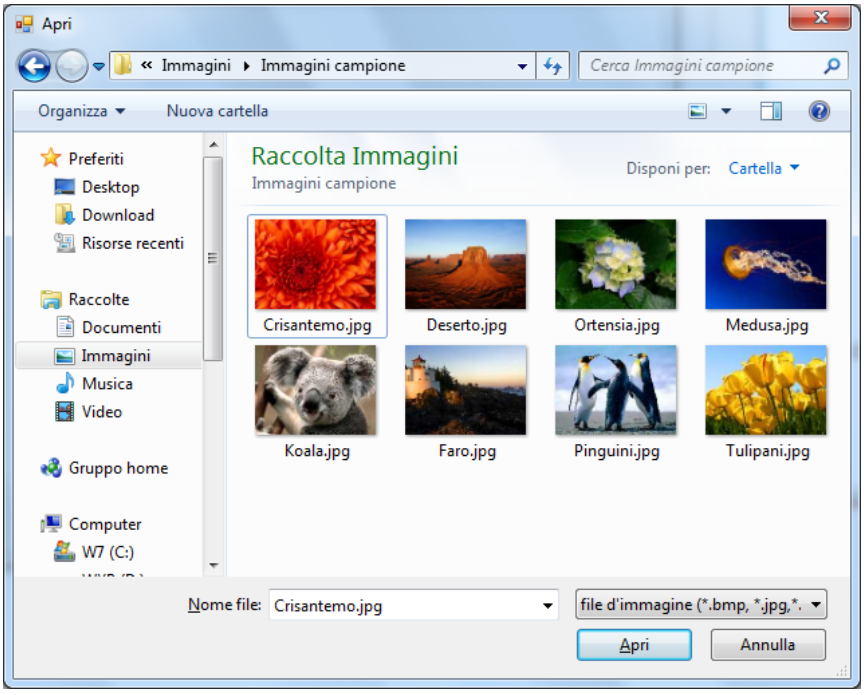



```
Private Sub btnApri_Click(…) Handles btnApri.Click
      Dim NomeFile As String, risp As Integer
      risp = dlgApri.ShowDialog()
      If risp = DialogResult.OK Then
          NomeFile = dlgApri.FileName
          picImmagine.Image = System.Drawing.Bitmap.FromFile(NomeFile)
      End If
End Sub
```

Questo è il valore dato alla proprietà Filter:

```
  file d'immagine (*.bmp, *.jpg,*.gif)|*.bmp;*.jpg;*.gif|Tutti i file (*.*)|*.*
```

Cliccando su Apri, compare la finestra di dialogo:



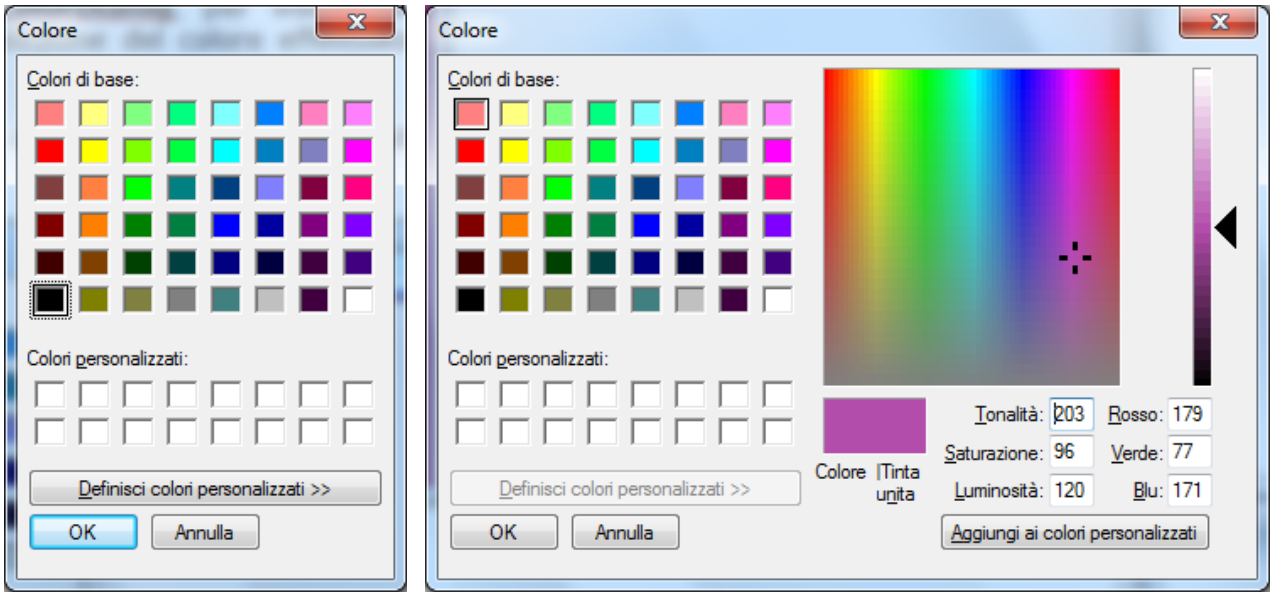
Cliccando su Apri, il nome del file viene assegnato alla proprietà **Image** del controllo **Picture** di nome picImmagine.

La finestra di dialogo Colore

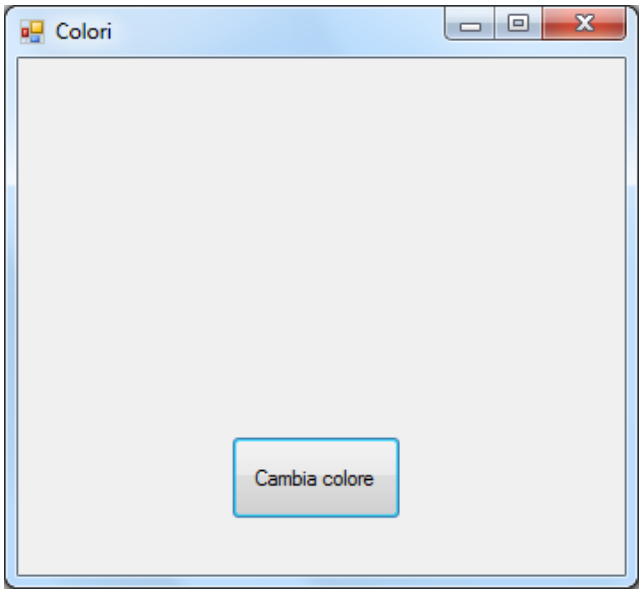
La finestra di dialogo Colore è una particolare finestra che permette all'utente di scegliere un colore da utilizzare nell'applicazione.

Una volta inserito il controllo **ColorDialog**, per visualizzare la finestra occorre utilizzare il metodo **ShowDialog**. Il risultato della selezione del colore effettuata dall'utente sarà assegnato alla proprietà **Color** del controllo.

La finestra Colore può apparire in due forme, una ridotta e l'altra estesa (che viene visualizzata cliccando sul pulsante **Definisci colori personalizzati**), in base al valore della proprietà **FullOpen**, che può essere **False** (ridotta) o **True** (estesa).



Esempio:

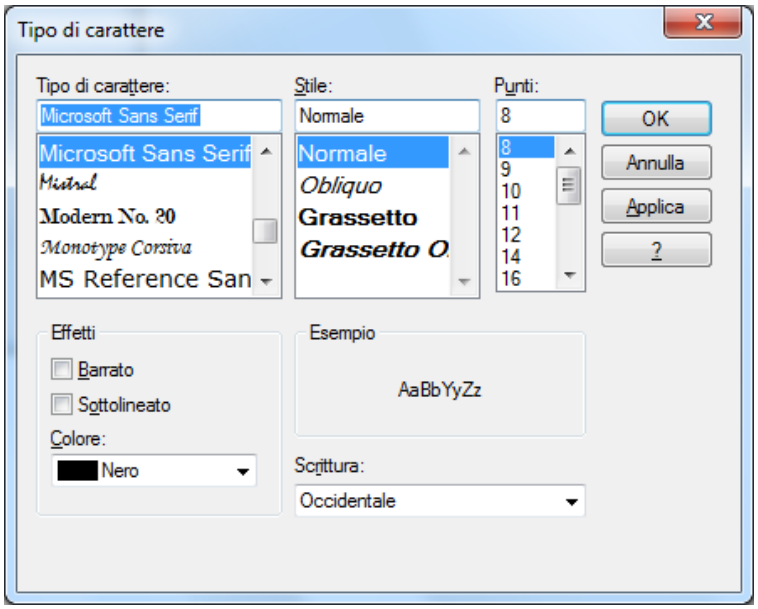



```
Private Sub btnCambiaColore_Click(…)
            Handles btnCambiaColore.Click

      dlgColore.ShowDialog()
      Me.BackColor = dlgColore.Color

End Sub
```

Quando si clicca sul pulsante, si fa apparire la finestra di dialogo e, se l'utente seleziona un colore, questo viene attribuito allo sfondo del form.

La finestra di dialogo Carattere

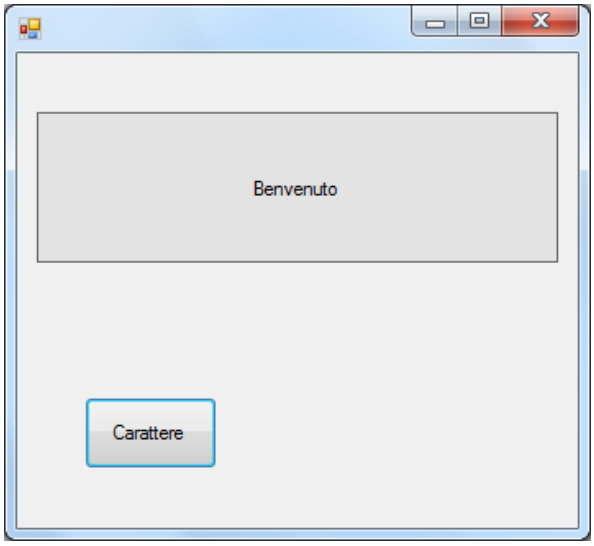
Permette all'utente di impostare, per il carattere del testo,
- il tipo (font),
- lo stile (Normale, Grassetto, Corsivo, Grassetto corsivo),
- le dimensioni (in punti)
- gli effetti (Barrato, Sottolineato)
- il colore.

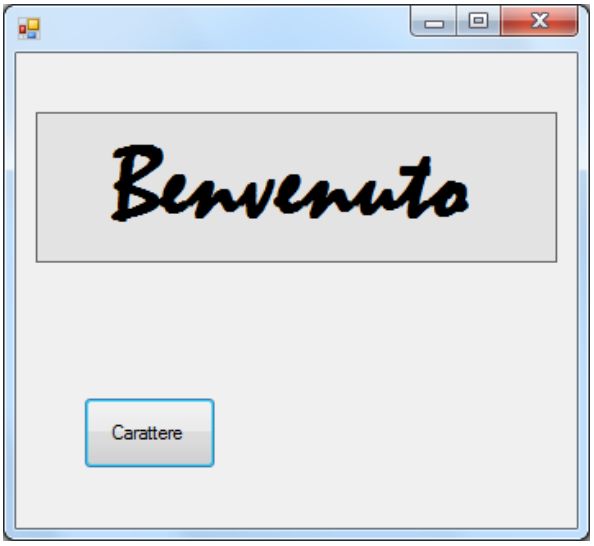


Il controllo da utilizzare è **FontDialog**. Le proprietà booleane **ShowApply**, **ShowColor**, **ShowEffects**, **ShowHelp** permettono di personalizzare la finestra, rendendo o meno visibili alcuni elementi (rispettivamente, il pulsante Applica, la scelta del colore del testo, la sezione Effetti, il pulsante di Help). Il controllo viene fatto apparire anche in questo caso con il metodo **ShowDialog**. Tutte le impostazioni effettuate dall'utente sono contenute nelle proprietà **Font** (che raccoglie le proprietà **Name**, **Size**, **Italic**, **Underline**, **Bold**, **Strikethru**) e **ForeColor**, che possono essere direttamente copiate nelle rispettive proprietà di un controllo di testo, come TextBox e Label.

Esempio:

Cliccando sul pulsante btnCarattere, appare la finestra di dialogo dlgCarattere, dove l'utente può dare le sue impostazioni. Dopo aver cliccato sul pulsante OK della finestra di dialogo, il testo nella label lblMessaggio subisce modifiche in base alle impostazioni date dall'utente.

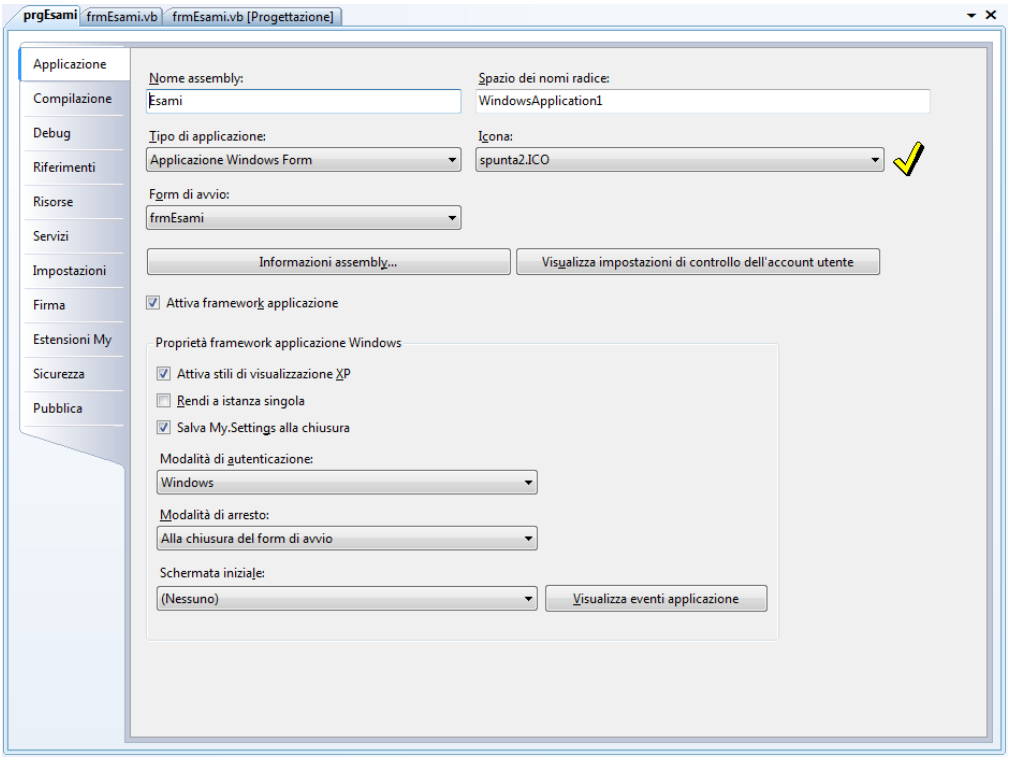





```
Private Sub btnCarattere_Click(…) Handles btnCarattere.Click
      dlgCarattere.ShowDialog()
      lblMessaggio.Font = dlgCarattere.Font
      lblMessaggio.ForeColor = dlgCarattere.Color
End Sub
```

Compilare e distribuire un'applicazione

Una volta terminate la realizzazione del programma e la necessaria fase di debugging, il programma è pronto per essere utilizzato. A questo scopo deve essere creata la sua versione eseguibile, cioè quella utilizzabile nell'ambiente di lavoro del Sistema Operativo. Si tratta di un file con estensione .exe. L'eseguibile è creato o aggiornato ad ogni compilazione e salvato nella sottocartella **bin** della cartella che contiene il progetto.
Per modificare le proprietà di un progetto, è necessario fare clic con il tasto destro del mouse sul nome del progetto stesso all'interno di **Esplora soluzioni** e selezionare **Proprietà**. Si aprirà una finestra di dialogo composta di diversi parti raggiungibili attraverso la vista ad albero sulla sinistra.



Nella sezione **Applicazione**, la casella di testo **Assembly name** permette di impostare il nome dell'eseguibile (senza estensione, che verrà aggiunta automaticamente); **Tipo di applicazione** specifica il tipo di applicazione che si sta creando e viene impostato quando si seleziona il tipo di progetto nella finestra New Project (Nuovo progetto); **Form di avvio** consente di definire il punto di ingresso dell'applicazione, ovvero il punto (ad esempio, il form) da cui inizierà l'esecuzione del programma quando lo avviamo; **Icona** permette di impostare l'icona dell'applicazione, quella predefinita è .

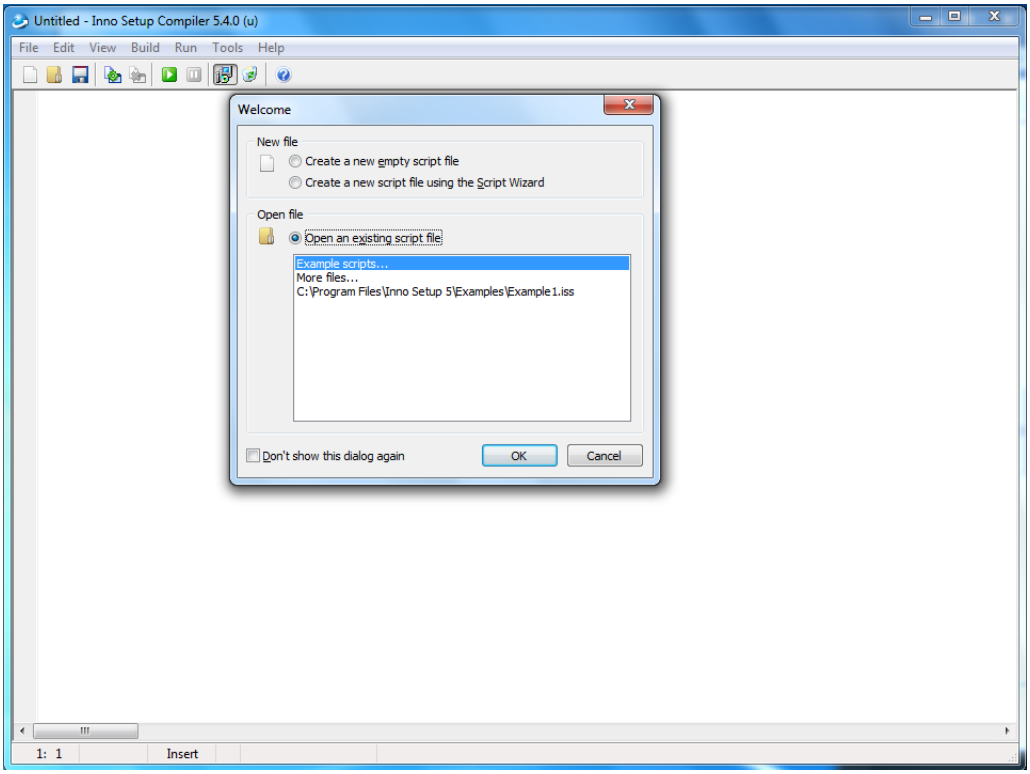
Visual Basic .NET introduce il concetto di **configurazioni di rilascio**, che determina il modo in cui un'applicazione viene compilata. Di default sono disponibili due opzioni: **Debug** e **Release**.
Usando la prima, oltre al programma vero e proprio, vengono generati, nella stessa cartella dell'eseguibile, più file che contengono informazioni di debug sul progetto. Sul programma, inoltre, non vengono effettuate ottimizzazioni, dal momento che esse renderebbero più difficile il debug stesso. Con la configurazione Release, invece, il programma è completamente ottimizzato. Un progetto compilato in configurazione Release genera dunque un eseguibile più piccolo di quello creato a partire dallo stesso progetto ma in modalità Debug.
È bene allora usare la configurazione Debug quando si sviluppa un'applicazione, per poi passare alla configurazione Release quando si genera l'eseguibile da distribuire. La prima è creata ad ogni esecuzione in debug del programma, la seconda selezionando l'opzione **Compila – Compila prgxxx** dal menu di VB. Ad essere distribuito sarà proprio il contenuto della cartella Release.

Attenzione: I programmi creati con Visual Basic. NET, per funzionare, necessitano del **runtime del framework .NET**: si tratta di un pacchetto di installazione delle dimensioni di circa 20 MB, fornito assieme a Visual Studio .NET e prelevabile anche sul sito Microsoft. È bene quindi informare l'utente e mettergli a disposizione, eventualmente, il runtime.

Creare un pacchetto d'installazione

Visual Basic Express non consente di creare un setup che faciliti all'utente l'operazione di installazione di un programma sul proprio computer, con link sul desktop e nel menu di avvio.
Si deve ricorrere allora ai cosiddetti **Installation builder**, programmi che permettono di realizzare pacchetti di installazione.

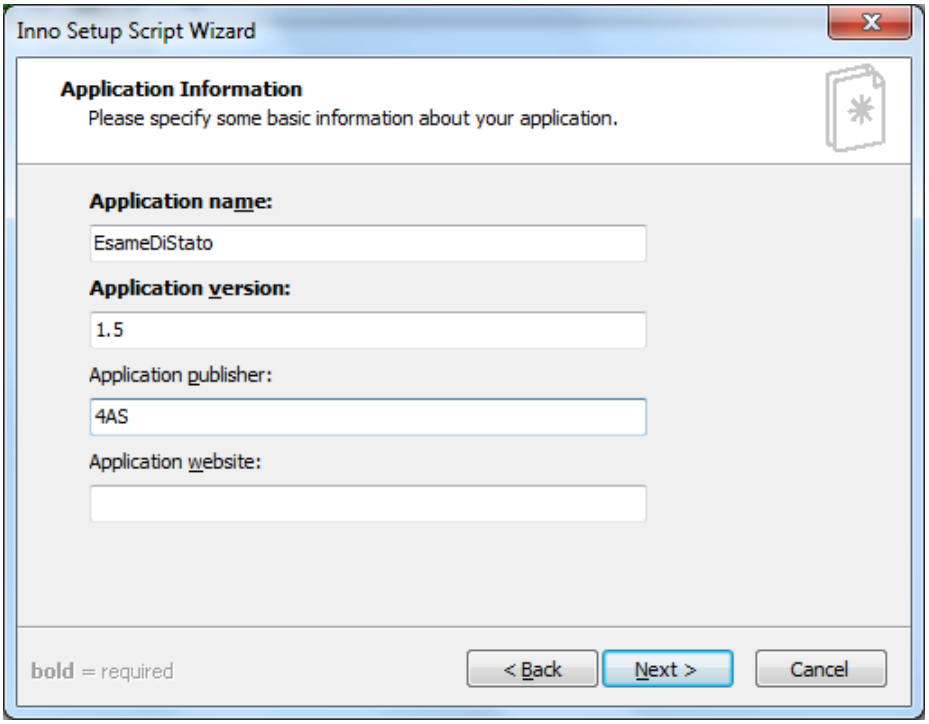
Uno di questi è il freeware **Inno Setup**. Avviando il programma, che si può scaricare gratuitamente da Internet, compare la seguente finestra.



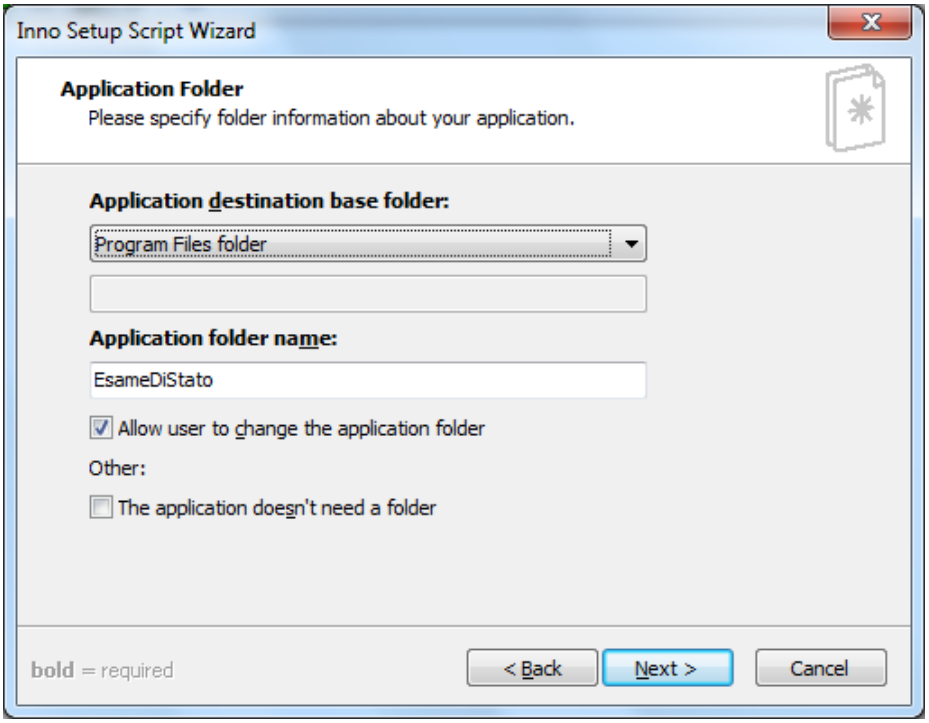
L'utente può formulare il pacchetto attraverso uno script in linguaggio specifico o utilizzare un semplice Wizard, una procedura guidata. Per attivare il wizard, dopo aver aperto il programma, cliccare su **Cancel** e selezionare **File** -> **New** dal menu (o premere **CTRL** + **N**):

Cliccare Next per proseguire.

Nella seconda finestra bisogna specificare:
- il nome dell'applicazione,
- l'indicatore di versione,
- la società che pubblica il software,
- il sito di riferimento per quel software.



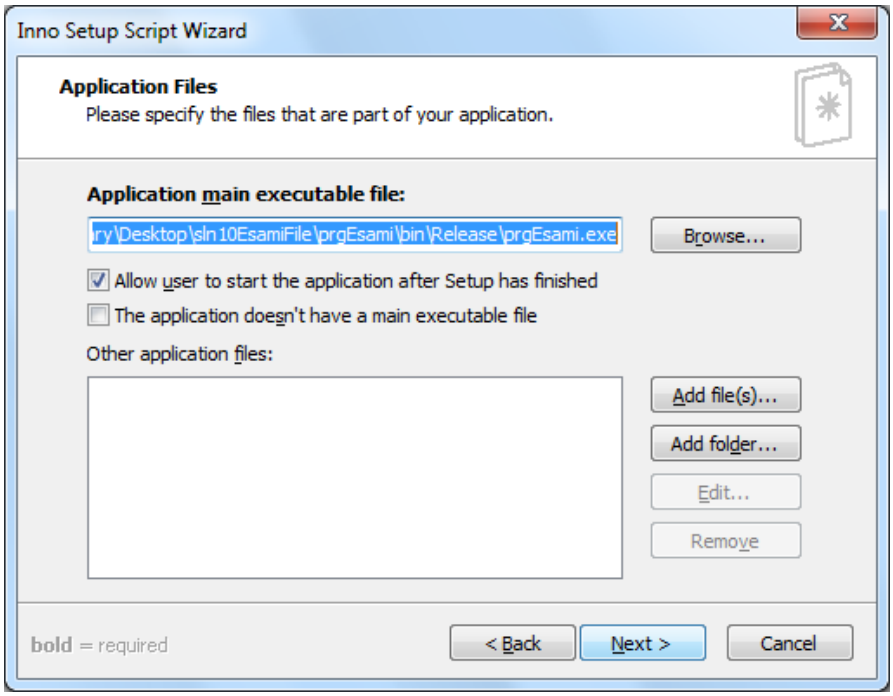
Nella terza finestra ci sono specifiche riguardanti la locazione d'installazione.



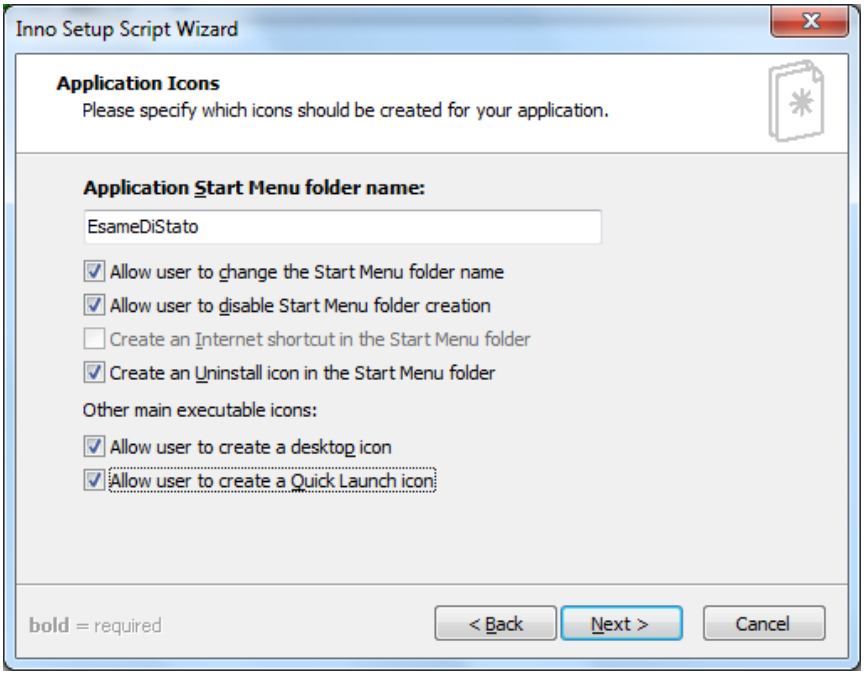
La ComboBox iniziale determina se l'applicazione sarà installata nella normale cartella Programmi

oppure in un altro percorso. Se si sceglie la seconda opzione ("Custom"), verrà sbloccata la TextBox sottostante nella quale inserire il percorso adatto.
La TextBox centrale indica il nome della cartella nella quale il programma viene installato. Di solito coincide con il nome dello stesso o al massimo con il nome della società
Le altre due checkbox indicano se lasciare all'utente la possibilità di cambiare la cartella oppure se il software non necessita di cartella. In quest'ultimo caso, si tratta di applicazioni con setup XCOPY, ossia delle quali basta una semplice copia sull'hard disk e niente di più per l'installazione.

Nella quarta finestra vengono richiesti i file da installare.



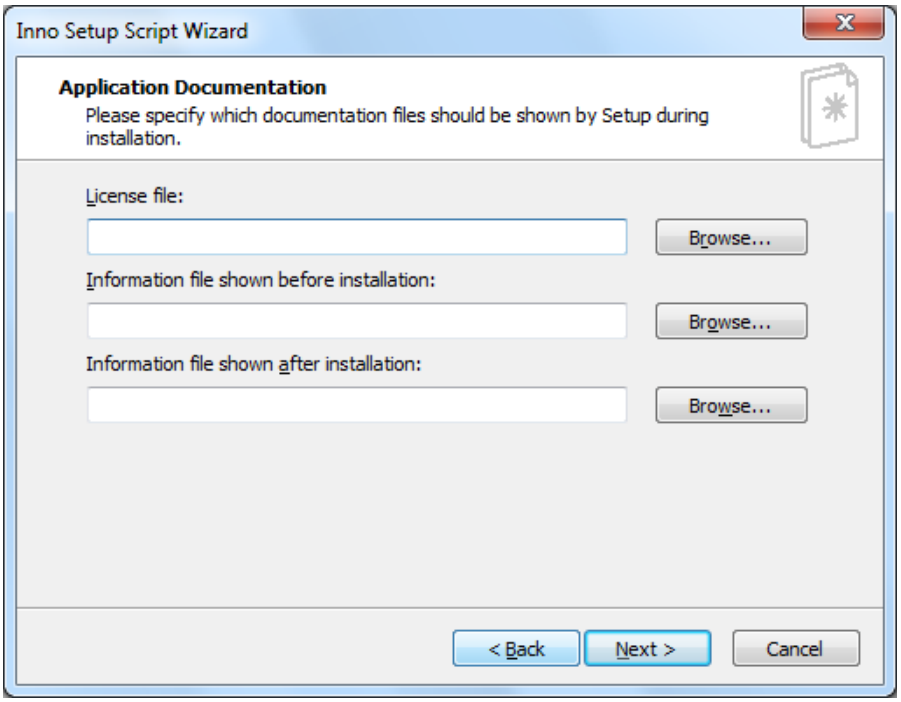
La prima TextBox richiede il percorso sull'hard disk dell'utente che sta scrivendo il setup dell'eseguibile principale, mentre la ListBox sottostante permette di aggiungere altri file o altre cartelle. Se si tratta di una cartella, non verrà copiato solo il suo contenuto, ma verrà anche creata la cartella stessa, comprensiva di ogni sottodirectory. Le CheckBox in mezzo determinano se si possa lanciare il software alla fine del setup e se non ci sia un eseguibile principale.
Nella quinta finestra si può decidere cosa deve comparire nel menu Start dopo l'installazione del programma ed eventuali altri link.



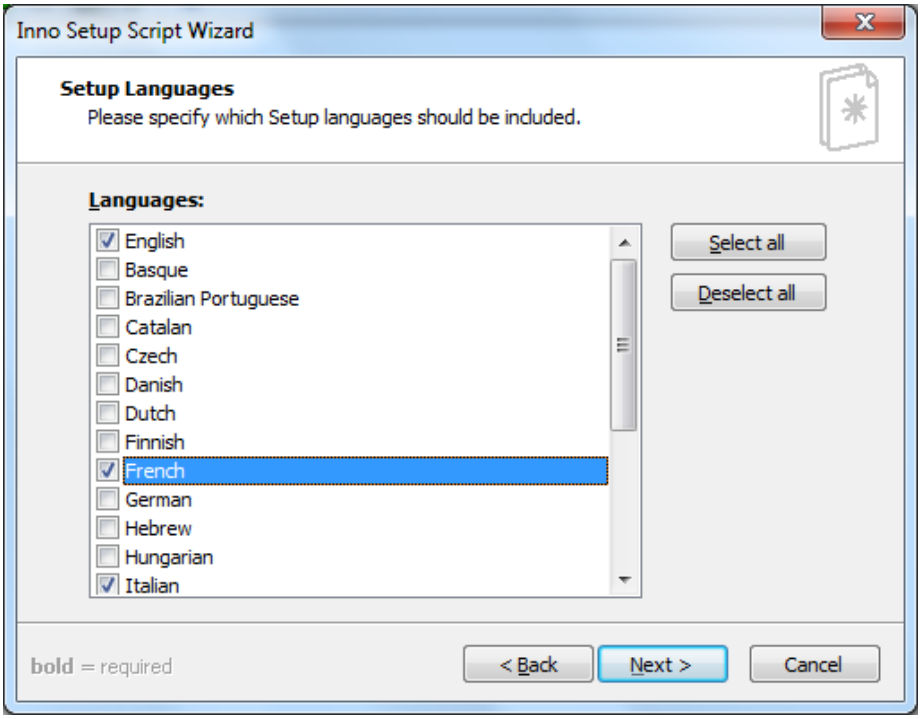
Indicare:

- il nome della cartella da creare nel menu "Tutti i programmi",
- se permettere all'utente, durante l'installazione, di modifica tale nome,
- se permettere all'utente, durante l'installazione, di non creare una cartella nel menu,
- se creare un collegamento al sito internet del software,
- se aggiungere l'opzione di disinstallazione,
- se creare un collegamento al programma di disinstallazione,
- se aggiungere un link sul desktop,
- se aggiungere un link nel menù Quick Launch.

La sesta finestra richiede di indicare il percorso del file della licenza, con le condizioni sotto le quali il software viene rilasciato, e di file da mostrare all'inizio e alla fine dell'installazione. Si tratta di file con estensione .txt spesso utilizzati, ma tutti opzionali.

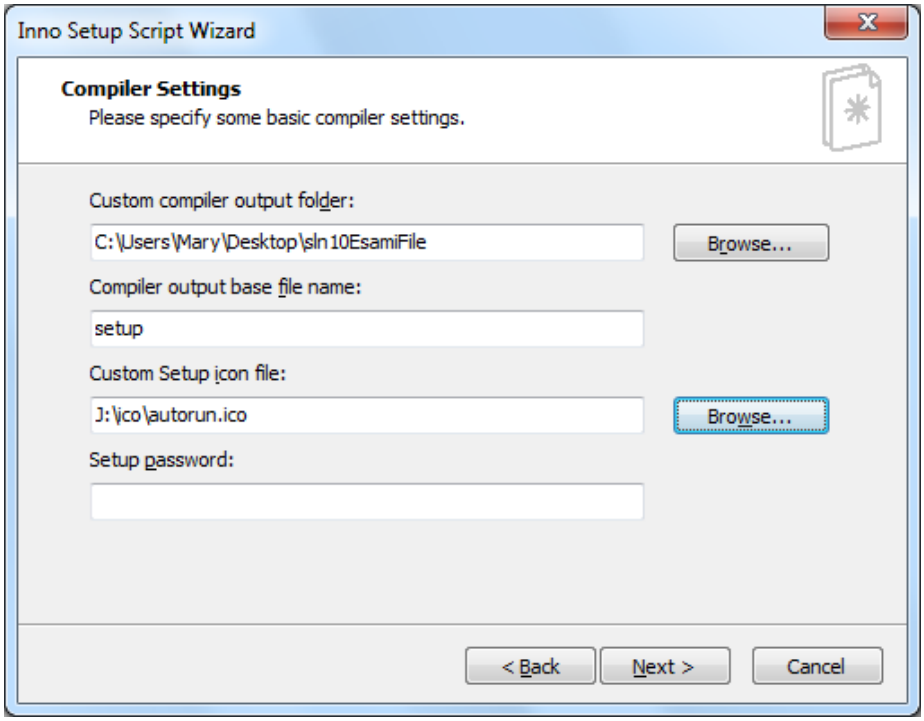


La settima finestra richiede la specifica delle lingue supportate per l'installazione:

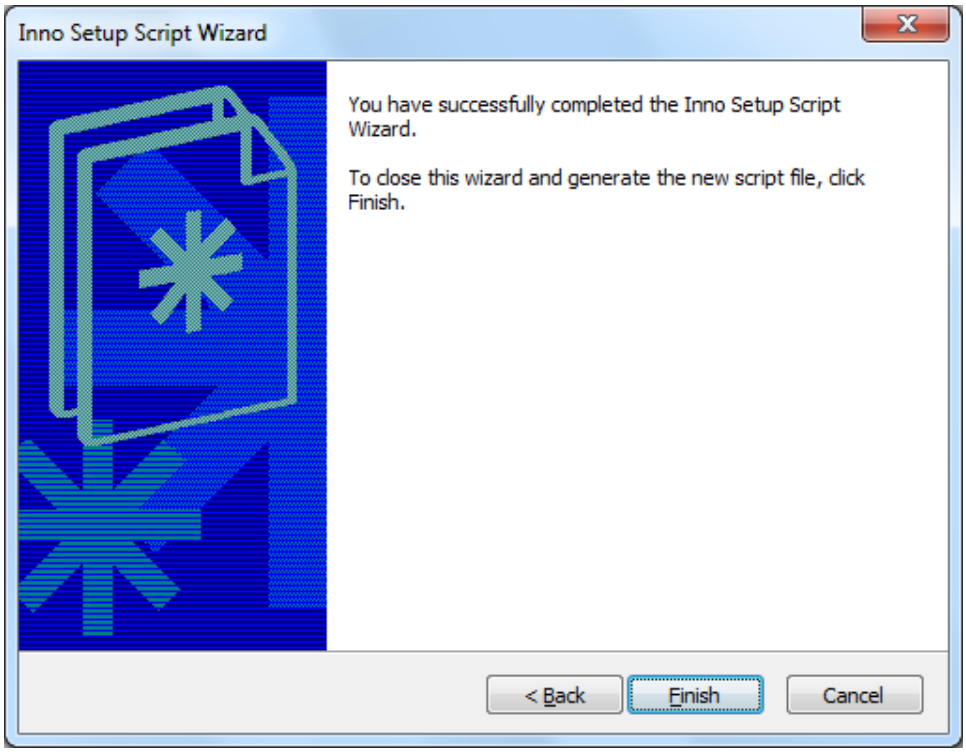


L'ottava finestra chiede di specificare:

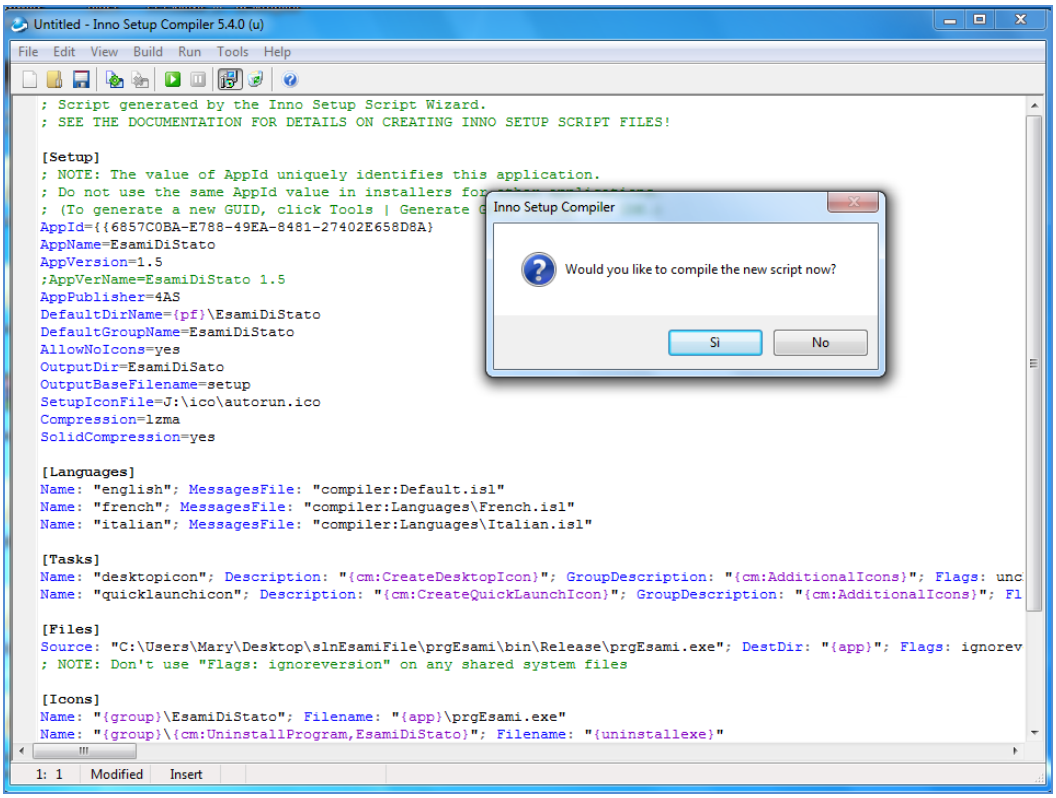
- la cartella dove creare il setup eseguibile completo,
- il nome dell'eseguibile (di solito Setup),
- un'icona per l'eseguibile (sono validi tutti i file *.ico),
- una password per accedere al setup.



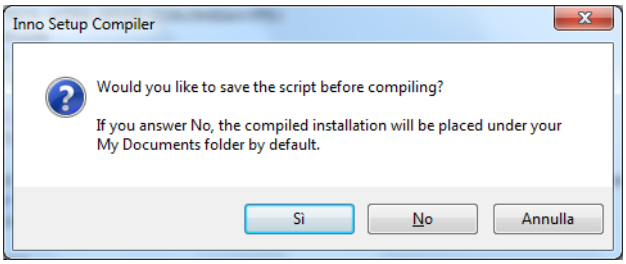
La nona finestra è quella conclusiva.

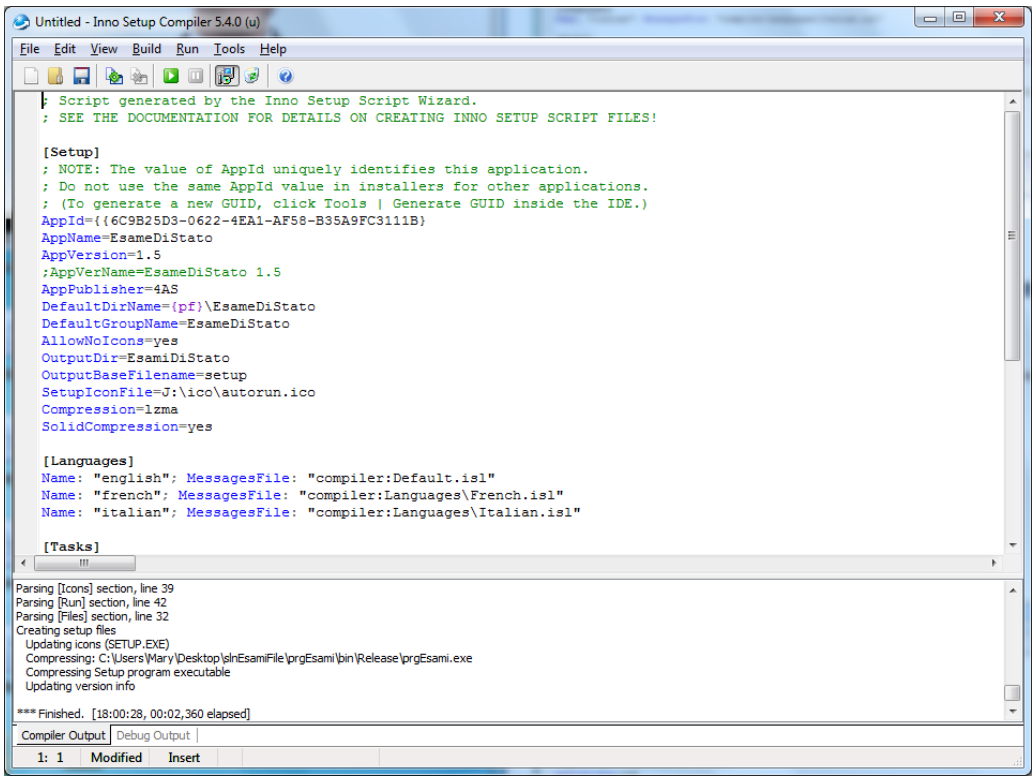


Una volta cliccato su **Finish**, appare nell'editor di testo di Inno Setup una serie di istruzioni, che compongono lo script del file di Setup. Inno Setup compilerà tali istruzioni per poi creare il pacchetto di installazione.
Subito dopo la fine del wizard verrà chiesto se eseguire la compilazione subito.



Cliccando Sì, sarà chiesto se e dove salvare lo script, un file con estensione .iss.



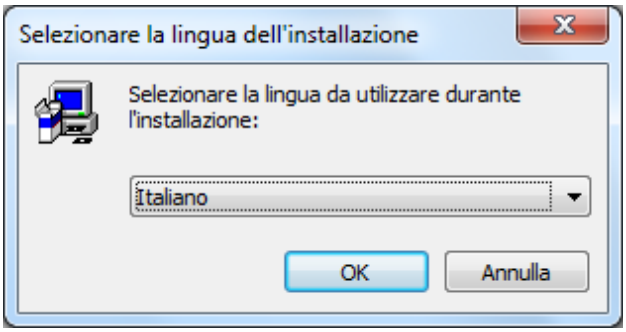


Ecco il file creato nella cartella specificata:

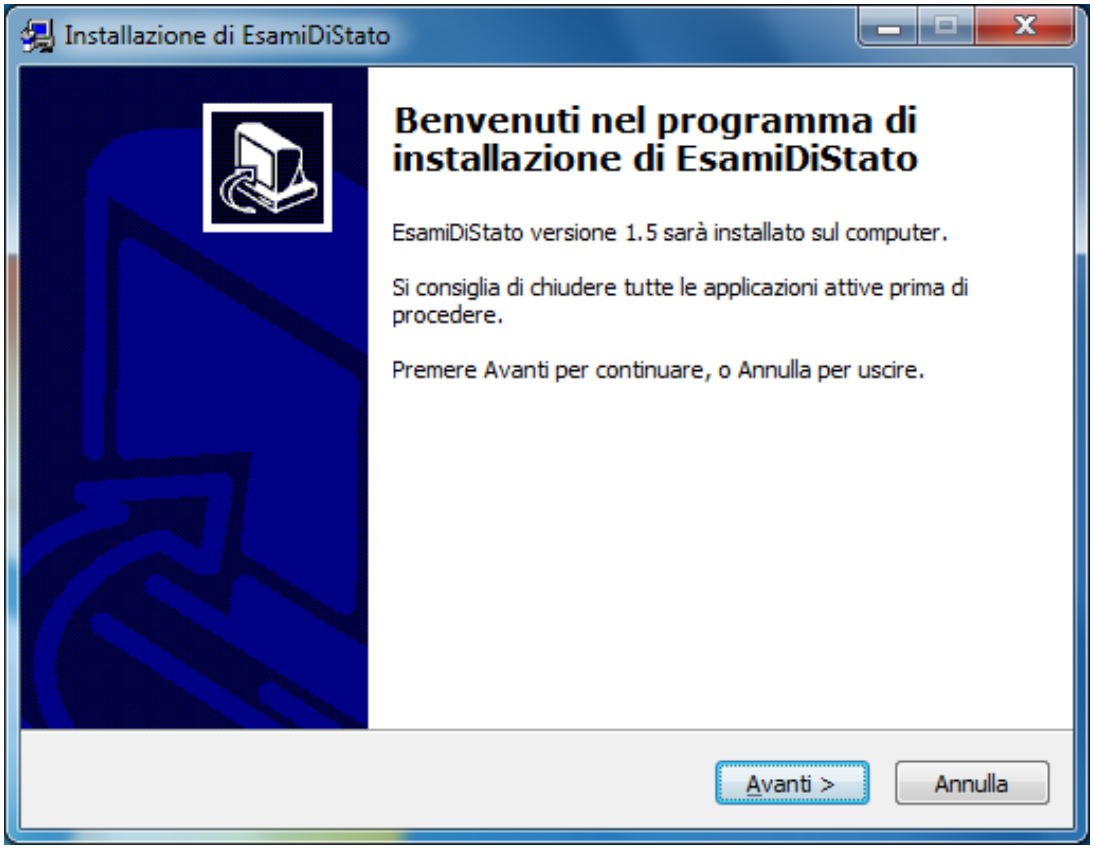
Installazione del programma

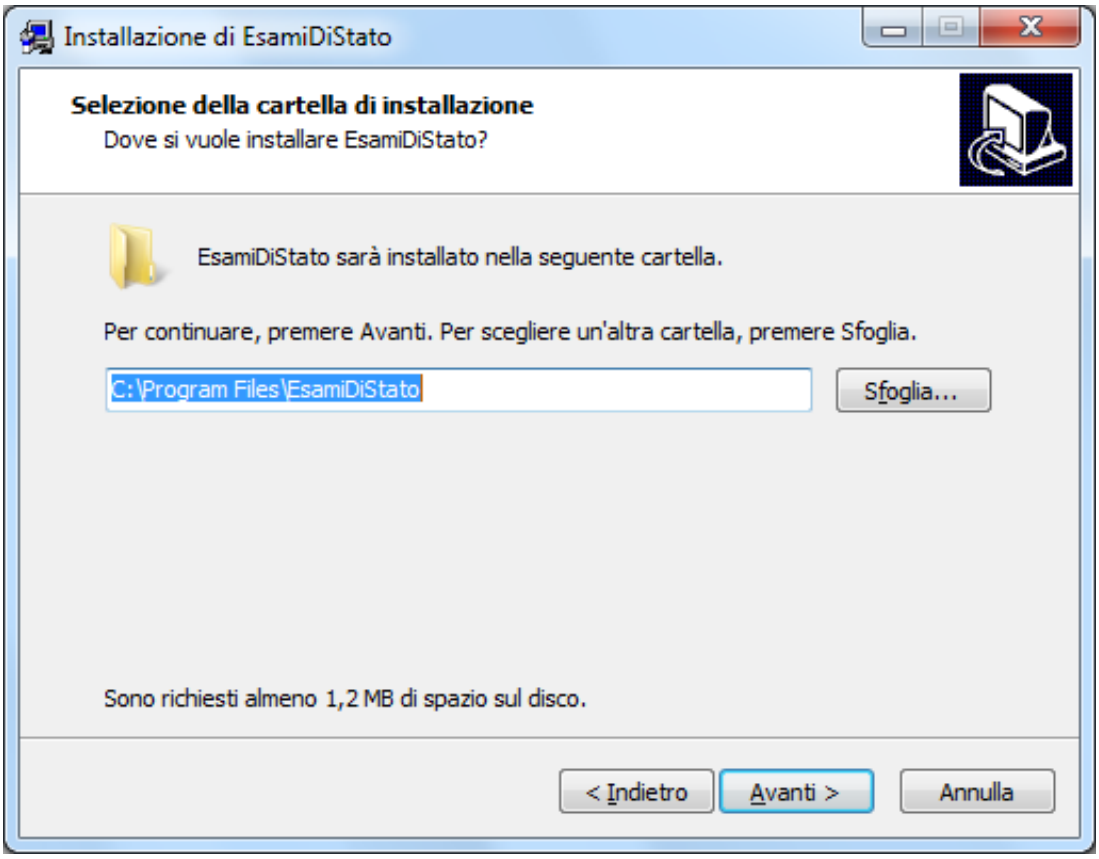
Cliccando due volte sul file setup.exe, parte la procedura di installazione.

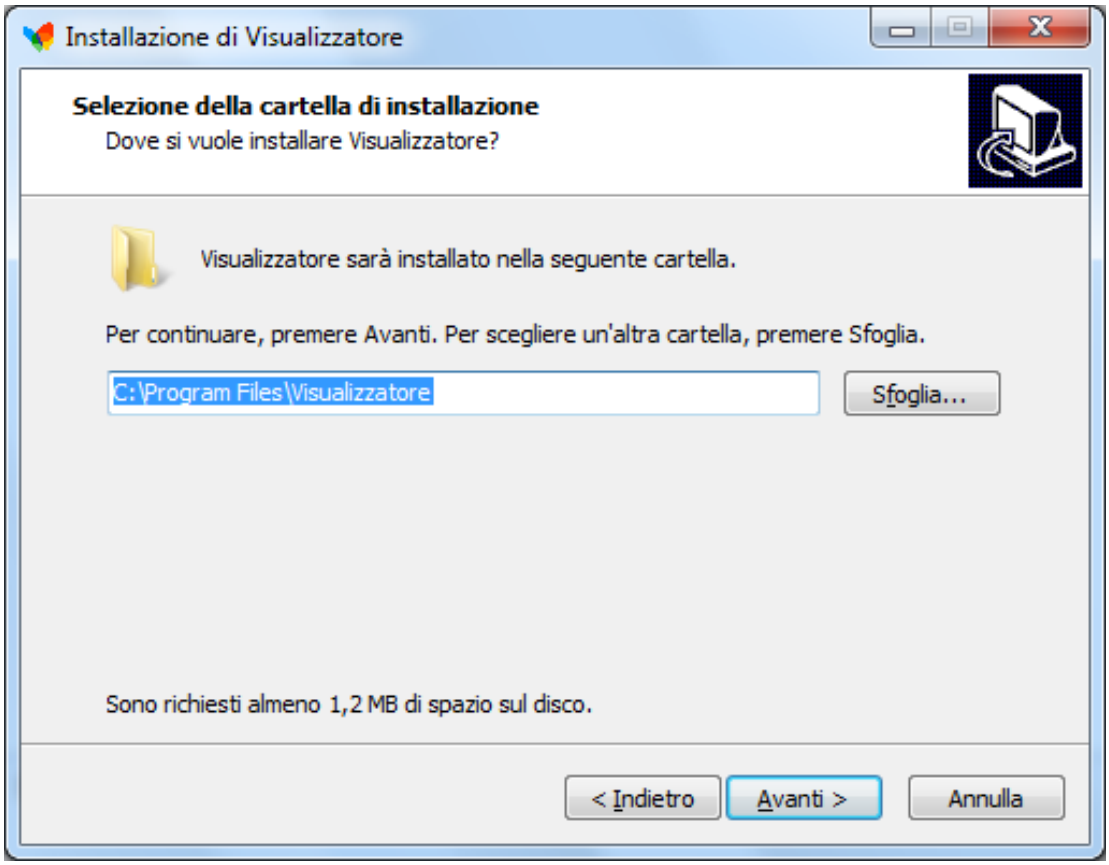
Compare subito la scelta della lingua, tra quelle indicate in fase di creazione del pacchetto.

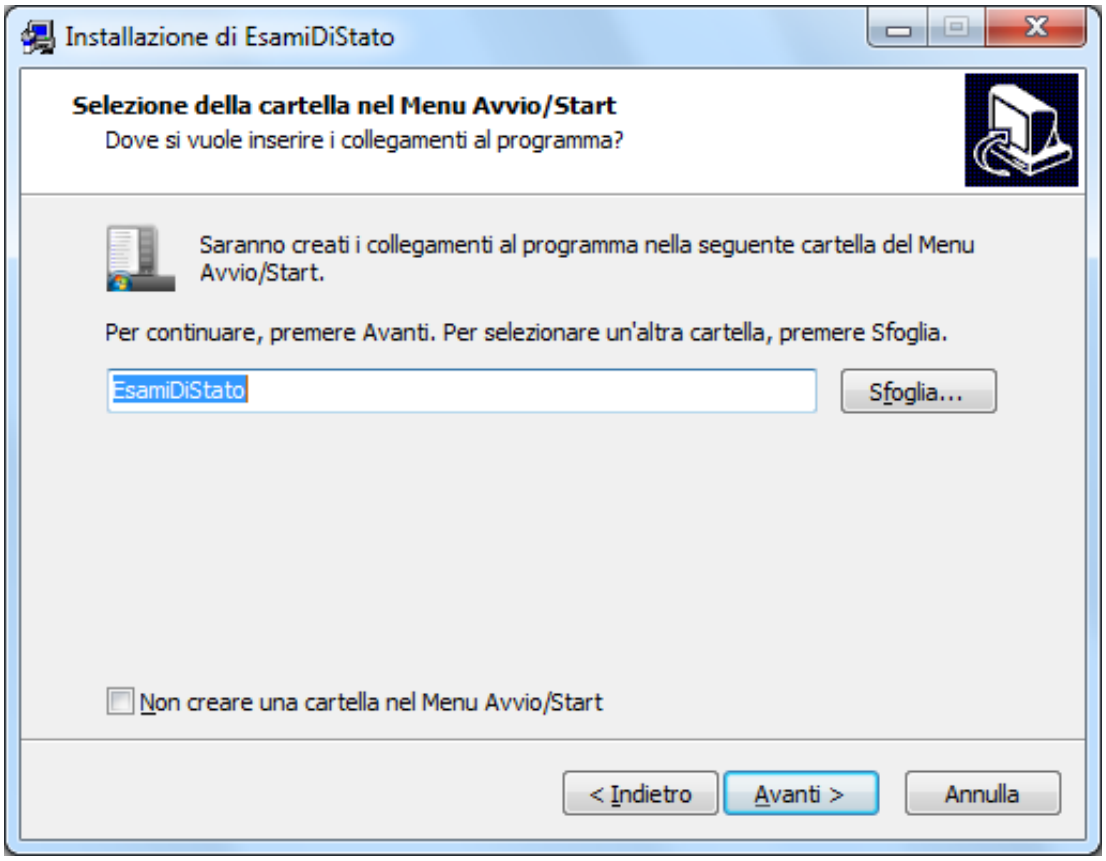


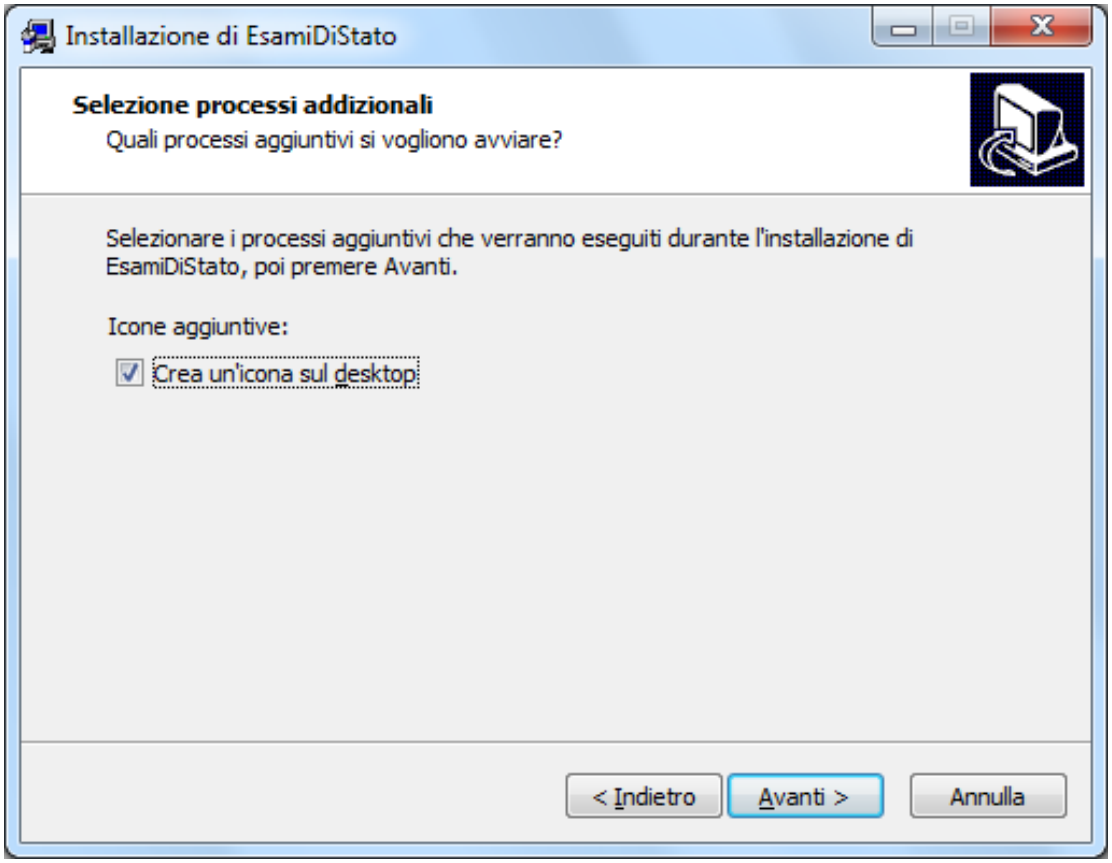
Cliccando su OK, inizia la fase di raccolta dei dati, in base alle specifiche date durante la creazione del pacchetto.

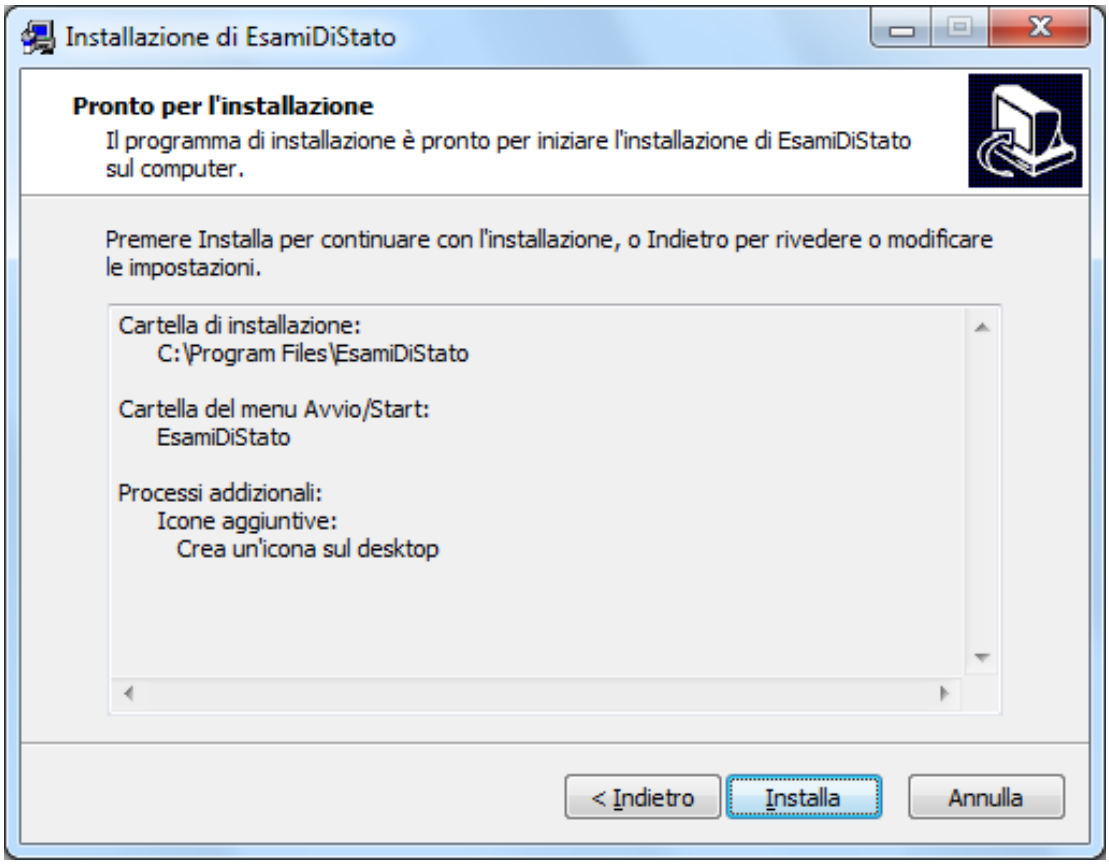



Installazione di EsamiDiStato
Selezione della cartella di installazione
Dove si vuole installare EsamiDiStato?
EsamiDiStato sarà installato nella seguente cartella.
Per continuare, premere Avanti. Per scegliere un'altra cartella, premere Sfoglia.
C:\Program Files\EsamiDiStato
Sfoglia...
Sono richiesti almeno 1,2 MB di spazio sul disco.
< Indietro
Avanti >
Annulla

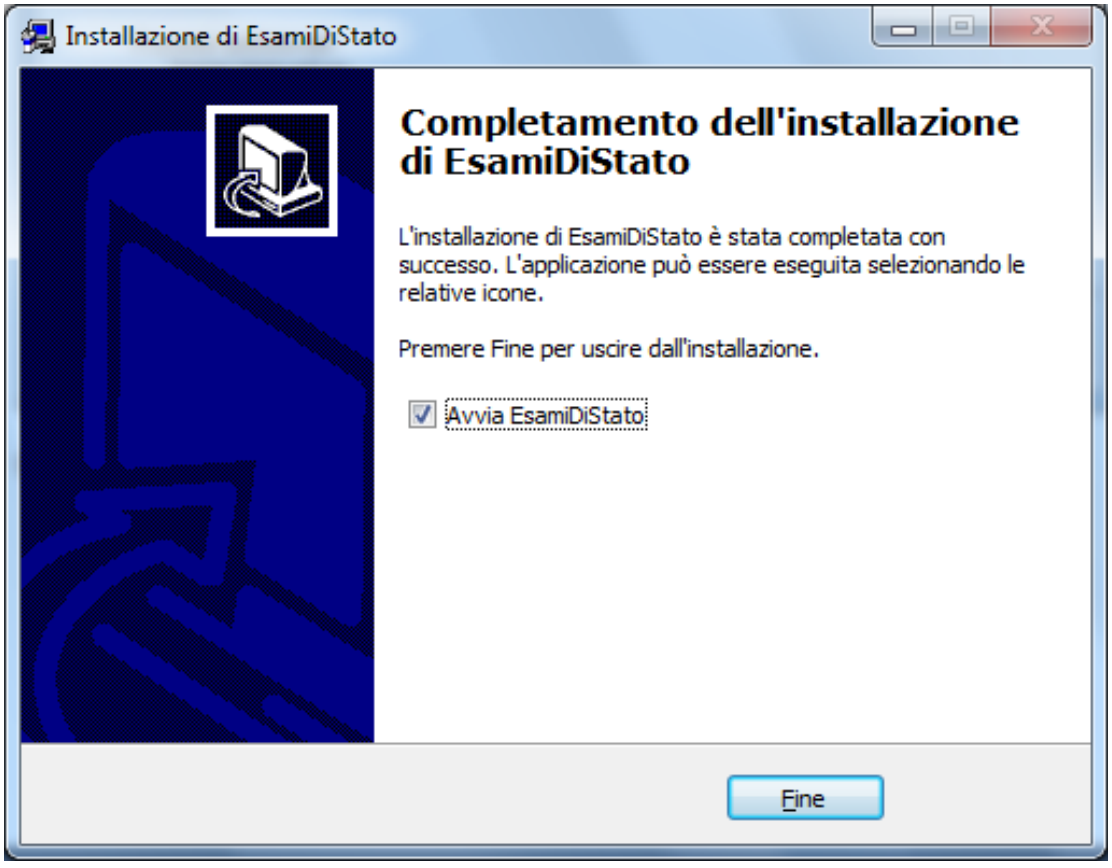

Installazione di Visualizzatore
Selezione della cartella di installazione
Dove si vuole installare Visualizzatore?
Visualizzatore sarà installato nella seguente cartella.
Per continuare, premere Avanti. Per scegliere un'altra cartella, premere Sfoglia.
C:\Program Files\Visualizzatore
Sfoglia...
Sono richiesti almeno 1,2 MB di spazio sul disco.
< Indietro
Avanti >
Annulla


Installazione di EsamiDiStato
Selezione della cartella nel Menu Avvio/Start
Dove si vuole inserire i collegamenti al programma?
Saranno creati i collegamenti al programma nella seguente cartella del Menu Avvio/Start.
Per continuare, premere Avanti. Per selezionare un'altra cartella, premere Sfoglia.
EsamiDiStato
Sfoglia...
Non creare una cartella nel Menu Avvio/Start
< Indietro
Avanti >
Annulla


Installazione di EsamiDiStato
Selezione processi addizionali
Quali processi aggiuntivi si vogliono avviare?
Selezionare i processi aggiuntivi che verranno eseguiti durante l'installazione di EsamiDiStato, poi premere Avanti.
Icone aggiuntive:
Crea un'icona sul desktop
< Indietro
Avanti >
Annulla



Cliccando su **Installa**, si avvia l'installazione.



Cliccando su Fine, il programma è installato e, se nell'ultima schermata è stata spuntata la casella di testo, il programma parte automaticamente, altrimenti lo si può avviare dal menu di avvio o dal desktop o dal Quick Launch, in base alle scelte effettuate.