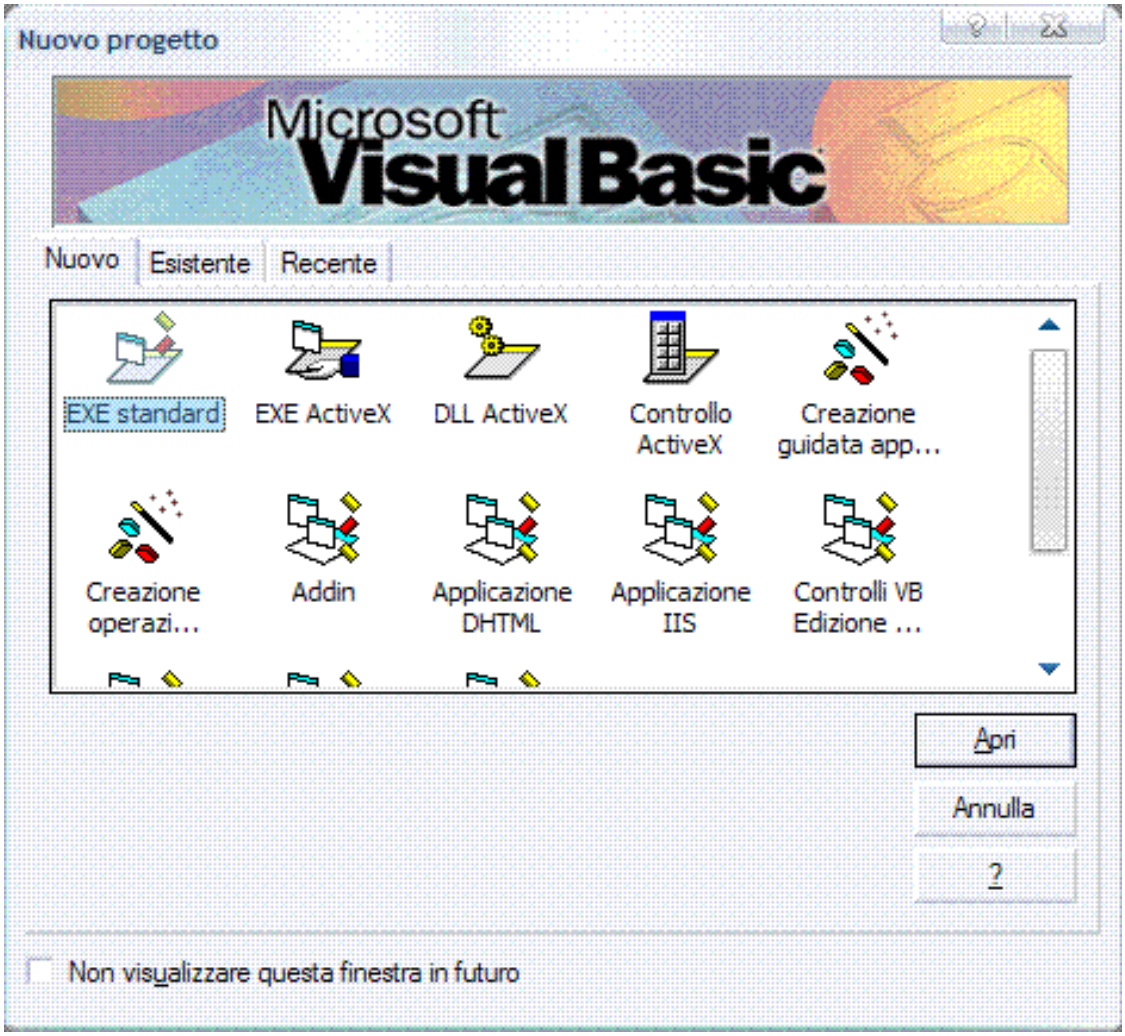
MASSIMO UBERTINI



VISUAL BASIC

WWW.UBERTINI.IT

VISUAL BASIC

INTRODUZIONE

Il linguaggio **BASIC** (*Beginner's All-purpose Symbolic Instruction Code*) è stato progettato al Darmouth College di Hanover (Ohio) nel 1964 da Thomas Kurtz e John Kemeny.
La Microsoft, legata per motivi storici ed affettivi al BASIC (1975 Paul Allen e Bill Gates sviluppano un inteprete BASIC), lo ha "riesumato" e lo ha integrato nella tecnologia Object Based realizzando Visual BASIC nel 1991, che comprende in pratica sia le classiche istruzioni e funzioni del BASIC (nelle sue numerose varianti: GWBASIC, BASICA, Quick BASIC 1981, Turbo BASIC) sia le istruzioni necessarie per gestire gli oggetti e le proprietà.
IDE (*Integrated Development Environment*) è l'interfaccia che Visual BASIC utilizza per consentire al programmatore di costruire le applicazioni.
È uno strumento rapido di sviluppo software **RAD** (*Rapid Application Development*).
Linguaggio di programmazione general purpose: top-down, programmazione strutturata, programmazione modulare, programmazione OO.
Linguaggio di script: **VBA** (*Visual BASIC for Applications*, per l'interazione con Microsoft Office), **VBScript** (*Visual BASIC Scripting Edition*, per l'interazione con il WWW).

TIPI DI DATI

Dim nomevariabile As tipo

- ✓ *Dim* è la parola chiave per tutte le variabili ed ha il significato di "Dimensiona", ovvero destina uno spazio di memoria per contenere il dato nome_variabile.
- ✓ *Nomevariabile* è il nome assegnato alla variabile.
- ✓ As è la parola chiave, che serve a stabilire il tipo di dato che deve essere immesso.
- ✓ *Tipo* indica la tipologia del dato da memorizzare.

È buona regola di programmazione utilizzare dei prefissi identificativi del tipo di variabile prima del nome arbitrario assegnato alla variabile stessa.
Per esempio, *Dim intpippo As Integer*.

1. Scalari

1.1. Standard

Data type	Size	Range
Byte	1 byte	0 a 255
Boolean	2 byte	True (1), False (0)
Integer%	2 byte	-32.768 a 32.767
Long&	4 byte	-2.147.483.648 a 2.147.483.647
Single!	4 byte	-3.4E+38 a -1.4E-45 | +1.4E-45 a +3.4E+38
Double#	8 byte -	-1.8E+308 a -4.9E-324 | +4.9E-324 a +1.8E+308
Currency@	8 byte	-922337203685477,5808 a +922337203685477,5807 fixed point, usato per calcoli contabili
Date	8 byte	dall'1/1/100 a 31/12/9999
Object	4 byte	qualsiasi riferimento Object
Variant	16 byte	default, è possibile memorizzare tutti i tipi di dati e le conversioni sono automatiche. I byte 0 e 1 contengono un numero che indica il tipo di dato memorizzato nei byte da 8 a 15. I byte da 2 a 7 non sono utilizzati. I byte da 8 a 15 contengono il valore effettivo della variabile.

1.2. Definiti

Il programamtore ha la necessità di lavorare su combinazioni di dati standard concatenati tra loro in modo logico.
In pratica, crea un tipo di dati contenente un certo numero di campi.

Data type **Size**

Type — Numero richiesto dagli elementi, questa definizione può avvenire solo nei moduli.

```
Type Studente
        Nome As String * 30
        Media As Integer
End Type
Dim Allievo As Studente
Allievo.Media = 8          'nomevariabile.nomecampo
Type Coordinata
        Ascissa As Single
        Ordinata As Single
End Type
Dim Punto As Coordinata
With Punto: .Ascissa = 5: .Ordinata = 7: End With
```

Enumerativo

```
Enum Festa
        sabato domenica
End Enum
Dim giorni as Festa
giorni = sabato
```

2. Strutturati

Data type	Size	Range
String$	1 byte per char	da 0 a 2E32 caratteri
Dim i As string		stringa a lunghezza variabile
i = "Via Garibaldi, 100"		i contiene 18 simboli
*Dim n As string * 20*		stringa a lunghezza fissa 20 caratteri
n = "Rossi Francesca"		n contiene 15 simboli e 5 spazi

Array statici: il numero di elementi è noto a compile time e non cambia a run time

Dim c (14) As Integer	array di 15 elementi (da 0 a 14)
Dim c (1 To 15) As Integer	array di 15 elementi (da 1 a 15)
Dim c (1 To 10,1 To 10) As Integer	matrice 10*10

Array dinamici: è possibile aggiungere o eliminare elementi a run time

```
Private Sub Command1_Click()
   Dim mat() As Integer              'matrice dinamica: dichiarazione
   ReDim mat(1 To 4, 1 To 4) As Integer' alloca nuovo spazio nella RAM
   Print: Print : Print "Dimensione matrice "
   Print "Righe da "; LBound(mat, 1); "  a"; UBound(mat, 1)
   Print "Colonne da "; LBound(mat, 2); "  a"; UBound(mat, 2)
'modifica l'ultima dimensione, se aumenta non cancella i dati
   ReDim Preserve mat(1 To 4, 1 To 2) As Integer
   Print "Preserve"
   Print "Righe da "; LBound(mat, 1); "  a"; UBound(mat, 1)
   Print "Colonne da "; LBound(mat, 2); "  a"; UBound(mat, 2)
End Sub
```

Record e File

Costanti

1. Le costanti *intrinseche* o *definite dal sistema* sono incorporate nei controlli, per esempio *vbTileHorizontal*.
2. Le costanti *simboliche* o *definite dall'utente* sono dichiarate utilizzando l'istruzione

[Public] [Private] Const nomecostante [As tipo] = espressione

Dichiarazioni: rintracciare la posizione
Per trovare l'istruzione con cui è stata dichiarata una variabile o una costante, selezionarle *Visualizza/Definizione (MAIUSC + F2)*

OPERATORI

Aritmetici	**Relazionali**	**Logici**
Potenza (^)	Uguale a (=)	Not
Meno unario (-)	Diverso da (<>)	And
Moltiplicazione, Divisione (* , /)	Minore di (<)	Or
Divisione intera (\ )	Maggiore di (>)	Ex-Or
Mod	Minore o Uguale a (<=)	Eqv (Ex-Nor)
Addizione, Sottrazione (+ , -)	Maggiore o Uguale a (>=)	Imp

Stringa concatenata (&, +) è meglio & perché fa la conversione automatica.

ISTRUZIONI

1. **Semplici** (non contengono altre istruzioni)
   Assegnazione, di chiamata, di salto (*Goto,Exit, On Error GoTo)*.
2. **Strutturate** (sono composte da più istruzioni)

SEQUENZA, le istruzioni sono eseguite nello stesso ordine in cui sono scritte;
SELEZIONE

unaria
```
If condizione Then istruzione
If condizione Then
        istruzioni
End If
```
binaria
```
If condizione Then
        istruzioni
Else
        istruzioni
End If
```
nidificata
```
If condizione1 Then
        [bloccoistruzioni1]
[ElseIf condizione2 Then
        [bloccoistruzioni2] ]
[Else
        [bloccoistruzioni-n] ]
End If
```
multipla (quando si hanno istruzioni mutuamente esclusive, altrimenti if nidificate)
```
Select Case espressione
        [Case espressioni1
                [bloccoistruzioni1] ]
        [Case espressioni2
                [bloccoistruzioni2] ]
        [Case Else
                [bloccoistruzioni-n] ]
End Select
```

ITERAZIONE successione di istruzioni eseguite ripetutamente
Il programmatore specifica la condizione che determina la fine dell'iterazione
ciclo a condizione iniziale
```
While condizione
        [bloccoistruzioni]
Wend                        itera per T esce per F
```

ciclo a condizione iniziale/finale

Do [{*While*} condizione]
[bloccoistruzioni]
[*Exit Do*]
[bloccoistruzioni]
Loop itera per T esce per F

Do [{*Until*} condizione]
[bloccoistruzioni]
[*Exit Do*]
[bloccoistruzioni]
Loop itera per F esce per T

Do
[bloccoistruzioni]
[*Exit Do*]
[bloccoistruzioni]
Loop [{*While*} condizione] itera per T esce per F

Do
[bloccoistruzioni]
[*Exit Do*]
[bloccoistruzioni]
Loop [{*Until*} condizione] itera per F esce per T

Il numero di iterazioni è noto a priori

For contatore = inizio *To* fine [*Step* incremento]
[bloccoistruzioni]
[*Exit For*]
[bloccoistruzioni]
Next contatore

For Each elemento *In* gruppo
istruzioni
Next elemento

Area di visibilità delle variabili

Variabili globali: sono variabili che sopravvivono per tutta la durata dell'applicazioine e la loro area di visibilità è l'intera applicazione.
Variabili locali: sono variabili legate all'esecuzione di una particolare routine e che sopravvivono solo il tempo necessario ad eseguire la routine.
Durata (**vita media**): periodo di tempo durante il quale la variabile considerata resta attiva e occupa memoria.
Area di visibilità: porzione di codice dalla quale è possibile accedere alla variabile considerata.
I moduli e i form possono contenere:

1. dichiarazioni: costanti, tipi, variabili, routine DLL;
2. routine, che in Visual BASIC si chiamano *Sub* e *Function*.

Identificatori utilizzati in un modulo o in un form.
È necessario dichiararli nella sezione dichiarazioni.

1. *[Private] Dim nomevariabile As tipo*: default, gli identificatori sono privati rispetto al modulo o alla form in cui compaiono, in altre parole sono disponibili per tutte le routine del modulo o della form.
2. *[Public] nomevariabile As tipo*: gli identificatori sono disponibili per tutti i moduli dell'applicazione, nel caso di identificatori uguali, è sufficiente indicare la form in cui sono contenuti Form1.Pippo, Form2.Pippo, Module1.Pippo.

Identificatori utilizzati in una routine

Sono riconosciuti solo nella routine per cui sono dichiarati: variabili locali.

1. *Dim nomevariabile As tipo*: default, gli identificatori sono privati e sono validi solo per la durata dell'esecuzione della routine.
2. *Static nomevariabile As tipo*: gli identificatori sono validi per la durata dell'esecuzione dell'applicazione. È possibile ottenere lo stesso risultato anche dichiarando la variabile nella sezione dichiarazioni del modulo o della form, in questo caso però la routine non avrebbe più l'accesso esclusivo a tale variabile.

```
Public Function somma(ByRef X As Integer, ByRef Y As Integer) As Integer
        Static cont As Integer
        cont = cont + 1
        MsgBox ("Chiamata numero " & Str(cont))
End Function
Private Sub Command1_Click()
        Dim Z As Integer
        Z = somma(3, 5)
        Z = somma(5, 7)
End Sub
```

Si osservi che possiamo passare variabili, ma anche valori costanti.

Le routine possono essere di tre tipi.

1. Sub

Eseguono un certo numero di operazioni, ma non ritorna nulla;

```
[Private|Public][Static] Sub momeroutine (argomenti)
                Dichiarazioni locali (Const, Dim)
                [bloccoistruzioni]
                [Exit Sub]
                [bloccoistruzioni]
End Sub
```

Esistono due tipi di Sub:

1. routine generali: istruzioni per eseguire particolari operazioni, devono essere richiamate in modo specifico con l'istruzione *Call Form1.nomeroutine (argomenti)*;
2. routine di eventi: quando è generato un evento, un oggetto richiama automaticamente la routine di eventi corrispondente; possono essere associate a controlli o a form.

La sintassi è la seguente

Sub nomecontrollo_nomeevnto, *Sub Form_nomeevento*.

2. Function

Calcolano un valore e lo restituiscono attraverso una notazione funzionale per essere usato direttamente nelle espressioni in modo simile alle variabili.

```
[Private|Public][Static] Function momeroutine (argomenti) [As tipo]
                Dichiarazioni locali (Const, Dim)
                [bloccoistruzioni]
                [Exit Function]
                [bloccoistruzioni]
                nomeroutine = ...        'assegno un valore al nome della funzione
End Function
```

3. Property

Restituiscono, assegnano valori ed impostano riferimenti ad oggetti.

```
[Private|Public][Static] Property momeproprietà (argomenti) [As tipo]
                [bloccoistruzioni]
End Property
```

Il meccanismo di sostituzione per cui ad un processo è possibile riapplicare argomenti diversi, flessibilità, in programmazione è chiamato: parametri, che sono di due tipi.

1. Parametri formali (PF): rappresentano gli oggetti su cui la routine opera, usati nel blocco, sono variabili locali.
2. Parametri attuali (PA): oggetti su cui opera effettivamente la routine all'atto della chiamata, sostituiscono i parametri formali in numero e tipo.

Il passaggio parametri avviene nei seguenti modi

- ✓ Per valore *(ByVal* argomento *As* tipo*)*: alla chiamata l'applicazione calcola il valore del PA, che è copiato nell'area di memoria del PF corrispondente e quindi locale alla routine, tutte le ulteriori operazioni avvengono sul PF. Allocando una nuova area di memoria si ha un'occupazione che potrebbe essere elevata se il PA dovesse essere strutturato e un consumo di tempo per le assegnazioni. È unidirezionale, eventuali assegnazioni durante il run non modificano il PA, al limite sporca l'ambiente locale, usato per dati in ingresso.
- ✓ Per indirizzo *(ByRef* argomento *As* tipo*):* è l'impostazione di default, alla chiamata l'indirizzo del PA è copiato nell'area di memoria del PF corrispondente, quando l'applicazione incontra il PF è sostituito con l'indirizzo del PA, in pratica si lavora sull'area di memoria del PA. Utile per i dati strutturati, ma pericolosa fonte di errori perché non esiste protezione e controllo. È bidirezionale, usato sia per l'ingresso dei dati sia per l'uscita dei risultati. I tipi di dati definiti dall'utente, i form ed i controlli possono essere passati solo per indirizzo, per esempio *(x As Control) (x As Form)*.

```
Public Function somma(ByRef X As Integer, ByVal Y As Integer) As Integer
        X = X + 1
        Y = Y + 1
        MsgBox ("Nella function X vale" & Str(X) & " e Y vale " & Str(Y))
        somma = X + Y
End Function
Private Sub Command1_Click()
        Dim A As Integer, B As Integer, Z As Integer
        A = 5
        B = 5
        MsgBox ("Prima di chiamare la function A vale" & Str(A) & " e B vale " & Str(B))
        Z = somma(A, B)
        MsgBox ("Dopo la chiamata A vale" & Str(A) & " e B vale " & Str(B))
End Sub
```

Ci deve essere concordanza tra PF e PA: in altre parole, rispettare sia il numero di parametri sia il tipo; se invece i parametri sono preceduti dalla clausola *Optional* argomento *As* tipo possono anche essere omessi al momento della chimata

```
Public Sub stampa(ByRef c As String, Optional ByRef n As String, Optional ByRef s As
String)
   Print: Print: Print c
   If Not IsMissing(n) Then Print n
   If Not IsMissing(s) Then Print s
End Sub
Private Sub Command1_Click()
   stampa "Verdi", "Mario", "Coniugato"
   stampa "Bianchi", "Sergio"
   stampa "Rossi"
End Sub
```

Ad un argomento facoltativo è possibile assegnare un **valore iniziale predefinito**.

```
Public Sub stampa(ByRef c As String, Optional ByRef n As String, Optional ByRef s As
```

```
String = "Celibe")
   Print: Print: Print c
   If Not IsMissing(n) Then Print n
   If Not IsMissing(s) Then Print s
End Sub
```

Numero arbitrario di argomenti *(ParamArray numeri())*

```
Public Function prodotto(ParamArray n() As Variant) As Single
   Dim e As Variant
   prodotto = 1
   For Each e In n
      prodotto = prodotto * e
   Next e
End Function
Private Sub Command1_Click()
   Print "Esempio 1  "; prodotto(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
   Print "Esempio 2   "; prodotto(12, 12)
End Sub
```

Per creare una routine generale è possibile:

1. scrivere la dichiarazione e le istruzioni all'interno della finestra (Codice);
2. *Strumenti/Inserisci routine...*

Per separare visivamente le routine:
Strumenti/Opzioni/Editor Impostazioni finestra Separa routine
Per visualizzare una sola routine per volta:
Strumenti/Opzioni/Editor Impostazioni finestra Visualizza modulo intero

LE FUNZIONI DI LIBRERIA

Funzione Abs	restituisce il valore assoluto di un numero
Funzione Atn	restituisce l'arcotangente di un numero
Funzione Cos	restituisce il coseno di un angolo
Funzione Exp	restituisce e (base dei logaritmi naturali) elevato a una potenza
Funzione Fix	restituisce la parte intera di un numero
Funzione Int	restituisce la parte intera di un numero
Funzione Round	arrotonda un numero decimale (3.5 = 4, 2.5 = 2!)
Funzione Log	restituisce il logaritmo naturale di un numero
Funzione Rnd	restituisce un numero casuale tra 0 (compreso) e 1 (escluso)
Funzione Sgn	restituisce il segno di un numero
Funzione Sin	restituisce il seno di un angolo
Funzione Sqr	restituisce la radice quadrata di un numero
Funzione Tan	restituisce la tangente di un angolo
Funzione Asc	restituisce il codice ASCII di un carattere
Funzione Chr	restituisce il carattere corrispondente a un certo codice ASCII
Funzione Cstr	converte in stringa
Funzione Val	restituisce il valore numerico di una stringa
Funzione Hex	converte in esadecimale
Funzione IsNumeric	verifica se un'espressione è numerica
Funzione Now	restituisce data e ora correnti
Funzione Time	restituisce l'ora corrente
Funzione Date	restituisce la data corrente
Funzione Timer	restituisce il numero di secondi dalla mezzanotte precedente
Funzione InStr	restituisce la prima occorrenza di una stringa all'interno di un'altra
Funzioni Left	restituisce i primi n caratteri a sinistra di una stringa
Funzione Right	restituisce gli ultimi n caratteri a destra di una stringa

Funzione Len restituisce la lunghezza della stringa in numero di caratteri
Funzione Mid restituisce un numero specifico di caratteri da una stringa
Funzione Instr restituisce la posizione di una stringa all'interno di un'altra
Funzione Lcase converte una stringa in minuscolo
Funzione Ucase converte una stringa in maiuscolo
Funzione Spc restituisce n spazi vuoti
Funzione Rtrim elimina gli spazi in coda ad una stringa
Funzione Ltrim elimina gli spazi in testa ad una stringa
Funzione Trim elimina gli spazi in testa e in coda ad una stringa
Funzione Str converte un numero in stringa
Funzione String restituisce una stringa contenente la ripetizione di un carattere n volte
Funzione Strcomp confronta due stringhe

Funzione equivalente derivata
Secante; Sec(X) = 1 / Cos(X)
Cosecante; Cosec(X) = 1 / Sin(X)
Cotangente; Cotan(X) = 1 / Tan(X)
Seno inverso; Arcsin(X) = Atn(X / Sqr(-X * X + 1))
Coseno inverso; Arccos(X) = Atn(-X / Sqr(-X * X + 1)) + 2 * Atn(1)
Secante inversa; Arcsec(X) = Atn(X / Sqr(X * X - 1)) + Sgn((X) -1) * (2 * Atn(1))
Cosecante inversa; Arccosec(X) = Atn(X / Sqr(X * X - 1)) + (Sgn(X) - 1) * (2 * Atn(1))
Cotangente inversa; Arccotan(X) = Atn(X) + 2 * Atn(1)
Seno iperbolico; HSin(X) = (Exp(X) - Exp(-X)) / 2
Coseno iperbolico; HCos(X) = (Exp(X) + Exp(-X)) / 2
Tangente iperbolica; HTan(X) = (Exp(X) - Exp(-X)) / (Exp(X) + Exp(-X))
Secante iperbolica; HSec(X) = 2 / (Exp(X) + Exp(-X))
Cosecante iperbolica; HCosec(X) = 2 / (Exp(X) - Exp(-X))
Cotangente iperbolica; HCotan(X) = (Exp(X) + Exp(-X)) / (Exp(X) - Exp(-X))
Seno iperbolico inverso; HArcsin(X) = Log(X + Sqr(X * X + 1))
Coseno iperbolico inverso; HArccos(X) = Log(X + Sqr(X * X - 1))
Tangente iperbolica inversa; HArctan(X) = Log((1 + X) / (1 - X)) / 2
Secante iperbolica inversa; HArcsec(X) = Log((Sqr(-X * X + 1) + 1) / X)
Cosecante iperbolica inversa; HArccosec(X) = Log((Sgn(X) * Sqr(X * X + 1) +1) / X)
Cotangente iperbolica inversa; HArccotan(X) = Log((X + 1) / (X - 1)) / 2
Logaritmo in base N; LogN(X) = Log(X) / Log(N)
Logaritmo decimale; LogDec (X) = Log (X) / 2.30258509

Funzione Format
Consente di convertire un valore numerico in una stringa di testo e di determinare quale sarà l'aspetto della stringa. Ad esempio, è possibile specificare il numero di cifre decimali, l'eventuale zero all'inizio o alla fine della stringa e i formati di valuta.
Format(*espressione*[, *formato*[, *primogiornosettimana*[, *primogiornoanno*]]])
L'argomento *espressione* specifica il numero da convertire, mentre l'argomento *formato* indica la stringa composta da simboli che rappresenta il formato per il numero.
Simbolo; **Descrizione**
- ✓ 0 segnaposto numerico, è sostituito da uno 0 all'inizio o alla fine di una stringa, se necessario.
- ✓ # segnaposto numerico; non è mai sostituito da uno 0 all'inizio o alla fine della stringa.
- ✓ . segnaposto decimale.
- ✓ ,; separatore delle migliaia.
- ✓ + $ ( ) spazio valore letterale; i caratteri sono visualizzati esattamente come digitati nella stringa del formato.

L'argomento *primogiornosettimana* è una costante che specifica il primo giorno della settimana, mentre l'argomento *primogiornoanno* è una costante che specifica il primo

giorno dell'anno. Entrambi gli argomenti sono facoltativi.

LE FINESTRE DI DIALOGO: INPUT/OUTPUT

Avviene in due modi.

1. **Casella di testo: TextBox (Input/Ouput)**
   È considerata una variabile, per cui si può utilizzare il suo contenuto. Il valore *Text* è di tipo *String*, se si deve effettuare la lettura di un numero bisogna convertirlo da *String* a valore numerico.
   Numero = Int (Casella_di_Testo.Text) *'Input*
   Casella_di_Testo.Text = Nome *'Output*
2. **Finestre di dialogo: InputBox (Input), MsgBox (Ouput)**
   Numero = Int (InputBox ("Inserisci il numero"))
   MsgBox (Nome, 0, "Il nome è")

La funzione InputBox visualizza un prompt attendendo che l'utente immetta una stringa d'input, quindi restituisce un valore corrispondente al contenuto.
RetString$ = InputBox$ (prompt$ [,titolo$ [,default$ [,x% [,y%]]]])
dove gli argomenti hanno il seguente significato:

RetString$	la stringa digitata dall'utente nella casella
prompt$	il tipo d'input atteso
default$	stringa di default, se l'utente non specifica una risposta diversa
x%, y%	posizione della casella in *twip* rispetto all'angolo superiore sinistro

La MsgBox visualizza una stringa e l'utente può rispondere solo tramite pulsanti, usata per l'output d'nformazioni (applicazione, autore) e messaggi di errore.
RetVal% = MsgBox (messaggio$ [,tipo [,titolo$ [helpfile]]])
dove gli argomenti hanno il seguente significato (solo il primo è obbligatorio, gli altri possono essere omessi usando le virgole):

messaggio$ che si vuole visualizzare, per più righe *VbCr* o Chr(13) *VbLf* o Chr(10)

tipo indica i pulsanti e/o le icone che si vogliono inserire nella casella:

0 *VbOKOnly*	solo il pulsante OK (default)
1 *VbOKCancel*	OK e Annulla
2 *VbAbortRetryIgnore*	Termina, Riprova, Ignora
3 *VbYesNoCancel*	Sì, No, Annulla
4 *VbYesNo*	Sì, No
5 *VbRetryCancel*	Riprova, Annulla
16 *VbCritical*	Stop
32 *VbQuestion*	punto interrogativo inscritto in un cerchio, (richiesta)
48 *VbExclamation*	punto esclamativo inscritto in un cerchio, (avvertimento)
64 *VbInformation*	"i" minuscola inscritta in un cerchio, (informazione)
0, (256, 512)	il primo (secondo, terzo) pulsante ha il focus per default

i valori possono essere sommati, 20 = 0 + 4 + 16 (OK, Sì, No, Stop)

titolo$ titolo della finestra, al massimo 255 caratteri

helpfile nome del file di help

RetVal% ritorna il pulsante premuto dall'utente:

1 *VbOK*	OK
2 *VbCancel*	Annulla
3 *VbAbort*	Termina
4 *VbRetry*	Riprova
5 *VbIgnore*	Ignora
6 *VbYes*	Sì
7 *VbNo*	No

La MsgBox può essere o funzione o istruzione, la differenza consiste nel fatto che la funzione ritorna un valore che indica quale pulsante l'utente ha premuto, l'istruzione no.

IMPOSTAZIONE DELLE OPZIONI

- ✓ *Strumenti/Opzioni/Editor Impostazioni codice Dichiarazioni di variabili obbligatoria* in questo modo bisogna dichiarare tutte le variabili prima del loro utilizzo, oppure scrivere all'inizio di ogni modulo l'istruzione *Option Explicit.*
- ✓ *Strumenti/Opzioni/Formato editor* per cambiare il colore e la dimensione del font.
- ✓ *Strumenti/Opzioni/Generale Impostazioni griglia form* per modificare la frequenza dei punti sulla griglia visibile della form.
- ✓ *Strumenti/Opzione/Ambiente All'avvio di un programma Salva con conferma* per salvare il progetto prima di eseguirlo

È possibile selezionare e gestire le aggiunte, ovvero estensioni che consentono di integrare l'ambiente di sviluppo aggiungendovi funzioni speciali.

- ✓ *Aggiunte/Gestore aggiunte/DataFormDesigner*
- ✓ *Aggiunte/gestore aggiunte/Visual BASIC Code Profiler*

GESTIONE PROGETTI

Quando si crea un'applicazione, è necessario utilizzare un file di progetto **VBP** (*Visual Basic Project* file di progetto), **VBG** (*Visual Basic Group* gruppo di progetti, **VBW** (*Visual Basic Window* lista delle finestre aperte) per gestire i diversi file.

- ✓ Form (**FRM**) è una finestra d'interfaccia per l'utente dell'applicazione; ciascun form visualizza controlli, immagini grafiche o altri form. Il modulo del form contiene il codice associato ai controlli, per gestire le risposte agli eventi provocate dall'utente o dal sistema.
- ✓ Form (**FRX**) un file di dati binario per tutti i form che includono controlli i cui valori sono dati binari.
- ✓ Classe (**CLS**) un file per ciascun modulo di classe, consente di definire tipi di oggetto aggiuntivi, non disponibili nelle classi esistenti, e di creare proprietà e metodi per tali oggetti; le proprietà e i metodi diventano membri della classe; la nuova classe può essere privata per l'applicazione o disponibile per altre applicazioni, programmazione OO.
- ✓ Modulo (**BAS**) un file per ciascun modulo standard, contiene le dichiarazioni globali, le costanti, le variabili e le routine della programmazione procedurale.
- ✓ Risorse (**RC**) un solo file di risorse, consente di raccogliere in una singola posizione tutto il testo e le bitmap specifici di una versione, file di risorse compilato (**RES**).
- ✓ Controlli utente (**CTL**) e dati binari di controlli utente(**CTX**)
- ✓ File pagine delle proprietà (**PAG**) e dati binari di pagine delle proprietà (**PGX**).
- ✓ Documento utente (**DOB**) e dati binari di un documento utente (**DOX**).
- ✓ Elenco delle dipendenze (**DEP**).
- ✓ File progettazione (**DSR**).
- ✓ Licenza di un controllo utente ActiveX (**VBL**).
- ✓ Estensioni di Visual BASIC (**VBX**).
- ✓ OLE Control eXtension (**OCX**) uno o più file contenenti controlli aggiuntivi, file di cache di un OCX (**OCA**).

Quando si creano, aggiungono o rimuovono file da un progetto, le modifiche apportate saranno visualizzate nella *Finestra Progetto* (Progetto1), contenente l'elenco corrente dei file del progetto: *Visualizza/Gestione progetti (CTRL+R).*

È possibile scegliere il pulsante "*Visualizza oggetto*" per visualizzare un form oppure il pulsante "*Visualizza codice"* per visualizzare il codice di un modulo standard, di classe o di form.

La finestra del codice contiene un modello per ciascuna routine di evento che può essere scritta; a destra c'è la casella "*Routine*", in cui sono elencati tutti gli eventi riconosciuti da Visual BASIC per il form o il controllo visualizzato nella casella "*Oggetto*".

Quando è selezionato un evento, nella finestra del codice è visualizzata la routine di eventi corrispondente; a sinistra c'è la casella "*Oggetto*", che consente di visualizzare il nome

dell'oggetto selezionato, fare clic sulla freccia a destra della casella di riepilogo per visualizzare l'elenco di tutti gli oggetti associati al form.
Se nella casella "*Oggetto*" è visualizzato "(*generale)*", nella casella "*Routine*" è visualizzato "(*dichiarazioni)*", fare clic sulla freccia a destra della casella di riepilogo per visualizzare l'elenco di tutte le dichiarazioni e le routine generali create per il form.

LA CASELLA DEGLI STRUMENTI

I controlli della casella degli strumenti rappresentano le classi. Quando si crea un controllo, è creata una copia o **Istanza** della classe di controllo.
I controlli sono strumenti, quali caselle, pulsanti ed etichette disegnate su un form per ricevere input o visualizzare output, e consentono inoltre di migliorare l'aspetto dei form.
La casella degli strumenti contiene un insieme di tools che è possibile usare per disegnare, muovere, o ridimensionare gli oggetti nel form.
Si può scegliere lo strumento voluto semplicemente cliccando su di esso.

- ✓ **Puntatore** questo è l'unico oggetto che non rappresenta in realtà un oggetto, si deve usare per ridimensionare o muovere un oggetto dopo che è stato disegnato nel form.
- ✓ **PictureBox** contiene un'immagine, che non è mai ridimensionata, può essere BMP, ICO o WMF nel form. Queste immagini possono essere decorative o attive.
- ✓ **Label** si usa per i testi che non si vuole siano modificati dall'utente, come per esempio un titolo sotto un grafico o una richiesta di input.
- ✓ **TextBox** si usa per permettere il cambiamento o l'inserimento di un testo all'utente.
- ✓ **Frame** si usa per creare un gruppo di oggetti funzionali, per raggrupparli, prima si disegna il controllo frame, poi gli oggetti all'interno del controllo.
- ✓ **CommandButton** si usa per creare un oggetto visibile che l'utente può selezionare per ottenere una certa funzione, un esempio è il classico pulsante di OK o FINE in un'applicazione.
- ✓ **CheckBox** si usa per visualizzare l'opzione vero o falso, o per mostrare scelte multiple quando è possibile scegliere più di una voce.
- ✓ **OptionButton** si usa in un gruppo per mostrare scelte multiple nelle quali l'utente può selezionare una sola voce.
- ✓ **ComboBox** si usa per disegnare una casella combinata di ListBox e TextBox. L'utente può selezionare qualcosa dalla lista o introdurre un valore nella TextBox.
- ✓ **ListBox** si usa per mostrare una casella di riepilogo attraverso la quale l'utente può selezionare qualcosa, la lista può essere visionata facendola scorrere a video se non è possibile visualizzarla interamente in una sola volta.
- ✓ **HScrollBar** si usa per permettere la navigazione veloce attraverso una lista di informazioni, per indicare la posizione corrente in una scala.
- ✓ **VScrollBar** si usa per permettere la navigazione veloce attraverso una lista di informazioni, per indicare la posizione corrente in una scala.
- ✓ **Timer** si usa per mettere degli intervalli di tempo definiti, è invisibile in esecuzione.
- ✓ **DriveListBox** si usa per mostrare il drive valido nel sistema in uso.
- ✓ **DirListBox** si usa per mostrare una lista gerarchica di directory.
- ✓ **FileListBox** si usa per mostrare una lista di file che l'utente può aprire, salvare o comunque manipolare.
- ✓ **Shape** consente d'inserire in un form rettangoli, quadrati, ellissi e cerchi.
- ✓ **Line** consente d'inserire in un form un segmento.
- ✓ **Image** contiene un'immagine che può essere BMP, ICO o WMF; a differenza dei CommandButton, non appaiano premuti quando si fa clic su di essi a meno che non si modifichi la bitmap con l'evento MouseDown. Se la proprietà *Stretch* è False, le dimensioni del controllo (*Width*, *Height*) si adattano automaticamente alle dimensioni del file immagine, al contrario sarà l'immagine ad adattarsi alle dimensioni del controllo.
- ✓ **Data** per creare applicazioni in grado di visualizzare e manipolare le tabelle di un database esistente.

- ✓ **OLE** consente al programmatore di creare applicazioni che possono usare dati provenienti da altre applicazioni.
- ✓ **CommonDialog** finestre di dialogo preimpostate, disponibili nel controllo CMDialog: Apri (1 ShowOpen), Salva con nome (2 ShowSave), Colori (3 ShowColor), Carattere (4 ShowFont), Stampa (5 ShowPrinter) Guida di Windows (6 ShowHelp).

I controlli sono suddivisi in tre categorie:

1. Controlli standard, sempre disponibili nella casella degli strumenti, si trovano nel file VB32.EXE.
2. Controlli aggiuntivi, si trovano nei file .OCX e possono essere rimossi o aggiunti, sono forniti dalla Microsoft o da altre case produttrici.
3. Oggetti inseribili OLE, ad esempio un foglio Excel.

Progetto/Componenti...(CTRL + T) /Scheda Controlli selezionarlo *Applica/Chiudi*

È buona regola di programmazione anteporre al nome del controllo:
- ✓ **frm** se si tratta di form (per esempio, frmInserimentoDati);
- ✓ **txt** per le caselle di testo (per esempio, txtNome);
- ✓ **lbl** per delle etichette (per esempio, lblCognome)
- ✓ **cmd** per i CommandButton (per esempio, cmdFine);
- ✓ **mnu** per i menu (per esempio mnuFile).

Creare una nuova scheda nella casella degli strumenti

Utile per organizzare i controlli secondo la tipologia.

1. Fare clic con il pulsante destro del mouse sulla casella degli strumenti.
2. *Aggiungi scheda/Nuovo nome della scheda/OK*.

CREAZIONE ED ESECUZIONE DI APPLICAZIONI

Durante la scrittura del codice è utile vedere tutte le proprietà ed i metodi disponibili per l'oggetto usato, dopo aver selezionato la proprietà la s'inserisce con la barra spaziatrice.
Modifica/Elenca proprietà/metodi
Per visualizzare la sintassi su un oggetto selezionato:
Modifica/Informazioni rapide
Per creare un'applicazione è necessario eseguire tre passaggi fondamentali:

1. Creare l'interfaccia utente aggiungendo ad un form i controlli desiderati, quali ad esempio caselle di testo e pulsanti di comando. Il form è una finestra che costituisce l'interfaccia della nostra applicazione. È possibile disegnare fino a 254 controlli in ogni form per creare un'applicazione che risponda pienamente alle nostre esigenze. Per creare una nuova form: *Progetto/Inserisci form*. Inserire una form di un altro progetto: *Progetto/Inserisci file...(CTRL + D)*. Eliminare una form *Progetto/Rimuovi Form1.* Per modificare la posizione dei controlli: *Formato/Allinea, Rendi uguale, Centra nel form*.
2. Impostare le proprietà del form e dei controlli per definire gli attributi con il comando *Visualizza/Proprietà F4*; per esempio, il titolo con la proprietà *Form1.Caption* si può modificare in "Prima applicazione"; si possono decidere anche i colori di foreground , di background, il colore e le dimensioni. Le proprietà sono impostate mediante la *Finestra Proprietà* (visualizza la classe ed un elenco delle impostazioni delle proprietà relative al form o al controllo selezionato).
3. Scrivere il codice per fornire la risposta agli eventi (il clic su un pulsante, l'apertura di un form o la digitazione in un campo) che si verificano nell'interfaccia, consentendo in questo modo l'interazione tra l'applicazione e l'utente. Utilizzare i metodi, in pratica le funzioni disponibili di ciascun controllo, per esempio Move per cambiare la posizione.
4. Si esegue l'applicazione tramite il menu *Esegui/Avvia F5*, s'interrompe con *Esegui/Fine*.
5. Salvare l'applicazione con *File/Salva progetto*.

Nella finestra di dialogo *Progetto/Proprietà di progetto1*... nella scheda Generale è

possibile specificare il Tipo di progetto, l'Oggetto di avvio, il Nome del progetto (quello memorizzato nel registro di Windows), il Nome file della Guida.
Nella scheda Crea è possibile specificare il Numero versione, il Titolo (nome assegnato all'applicazione), l'Icona che identifica l'applicazione, le Informazioni sulla versione (Commenti, Copyright, Descrizione del file, Marchi registrati, Nome prodotto, Nome società).
Nella scheda Compila se si sceglie Compila in **P-Code** si ottiene un'applicazione che occupa poco spazio, ma necessita del Runtime (MSVBVM60.DLL) per essere eseguita (non sceglierla); Compila in **codice nativo** genera un eseguibile vero e proprio, inoltre è possibile ottimizzare in velocità (occupa più spazio), in dimensione (è più lento), non ottimizzare oppure ottimizzazioni avanzate (per programmatori esperti).
File/Crea Progetto1.exe... si salva come eseguibile, ma per funzionare correttamente bisogna inserire gli OCX usati nel progetto, la ricerca avviene seconda questa priorità.
1. La cartella dell'eseguibile.
2. C:\Windows\system32.

Per vedere tutte le dipendenze dell'eseguibile si usa: *depends.exe.*
È possibile utilizzare la **compilazione condizionale**.
Costruiamo la "Seconda applicazione", disegnare il form seguente:
1. inserire una TextBox (casella di testo);
2. inserire un CommandButton (pulsante di comando).

Quando si fa clic sul pulsante si deve visualizzare nella casella di testo "Hello, World". Bisogna, quindi, collegare il codice all'evento appropriato, per fare questo occorre:

Oggetto	Proprietà	Impostazione
Form1	Caption	Seconda applicazione
	BorderStyle	1 (la finestra non può essere ridimensionata)
Tex1	Text	""
Command1	Caption	OK
	Default	True (predefinito, quando si preme CR)
	TabStop	True (evidenziato)

Scrivere il seguente codice, doppio clic su Command1 per aprire la finestra di codice:

```
Private Sub Command1_Click()
    Text1.Text = "Hello, world"
End Sub
```

Command1 è il nome dell'oggetto al quale si vuole associare il codice.
Click è il nome dell'evento, ossia l'azione che, se attuata, permette l'esecuzione di una certa istruzione (in questo specifico caso si riferisce al clic con il mouse).
Text1.Text è l'istruzione, ossia il codice che si vuole eseguire quando l'evento occorre.

Operazione	Metodo
Caricamento di un form in memoria senza visualizzarlo	*Load*
Caricamento e visualizzazione	*Show*
Chiusura di un form	*Hide*
Chiusura di un form e scaricamento dalla memoria	*Unload*
È visualizzato o no sulla barra delle applicazioni	*ShowInTaskbar*

Per terminare in modo corretto un'applicazione si deve usare una o più'istruzioni *Unload* che scaricano dalla memoria tutti i form del progetto. La chiusura di un'applicazione soltanto con *End* non è efficiente perché, anche se termina l'esecuzione, mantiene i form allocati nella memoria.

Per determinare gli eventi per i quali scrivere il codice, è necessario prevedere le azioni dell'utente e la risposta che si desidera ottenere dall'applicazione.

Le routine di eventi consentono inoltre di:
- ✓ generare altre routine di eventi;
- ✓ modificare le proprietà di un oggetto;
- ✓ richiamare altre routine generali non collegate ad un evento particolare.

In conclusione, il funzionamento di un'applicazione orientata agli eventi è il seguente.
1. L'applicazione è avviata ed il form di avvio è caricato e visualizzato automaticamente; se è necessario eseguire determinate operazioni, queste devono essere specificate nella routine di eventi *Form_Load(). Form_Unload(Cancel As Integer)* l'evento è generato prima che il form sia chiuso, il valore integer è passato al sistema operativo.
2. Un form o un controllo riceve un evento che può essere provocato dall'utente, dal sistema o dal codice.
3. Se è prevista la routine di eventi corrispondente, la routine sarà eseguita.
4. L'applicazione attende l'evento successivo.

Nella programmazione visuale ogni controllo (oggetto) è rappresentato da proprietà, ma anche da metodi (vere e proprie routine che il controllo è in grado di eseguire). All'inserimento del punto dopo il nome dell'oggetto, compaiono tutte le proprietà (icona verde) ed i metodi dell'oggetto stesso in una finestra a comparsa.

Per esempio, nel caso di un form i metodi principali sono.
Hide: permette di nascondere un form senza scaricarlo dalla memoria: maggiore velocità!
Show: permette di visualizzare un form.
Move: permette di muovere un form ridimensionandolo.

```
Private Sub CmdRidimensiona_Click()
        Me.Move Me.Left, Me.Top, Text1.Text, Text2.Text
End Sub
Private Sub CmdSposta_Click()
        Me.Move Text1.Text, Text2.Text, Me.Width, Me.Height
End Sub
```

In questo caso al metodo Move sono passati i valori contenuti nelle due caselle di testo, a parametri diversi a seconda se si utilizza il tasto Ridimensiona o Sposta.
È utile usare il *Visualizza/Visualizzatore oggetti...(F2)*., scegliendo il nostro progetto nella casella in alto a sinistra consente di visualizzare gli oggetti disponibili nelle librerie e nel progetto e le relative proprietà e metodi, risulta poi semplice individuare le routine che c'interessano (doppio clic per aprire il codice).
Le informazioni sono visualizzate su tre livelli:
1. Librerie/Progetti;
2. Classi/Moduli;
3. Metodi/Proprietà.

Elaborazione in background

La funzione di nome *DoEvents()*, quando è eseguita, permette al sistema operativo di riprendere il controllo della CPU per far proseguire l'esecuzione delle altre applicazioni che hanno eventi in sospeso.
Utile all'interno di cicli molto lunghi quali la scansione di un disco fisso o loop di calcoli matematici. In pratica, non comporta la disattivazione dell'applicazione corrente, ma abilita l'elaborazione degli eventi in background.

Tecniche di scrittura del codice

- ✓ Un'istruzione per riga, è però possibile inserire più istruzioni sulla stessa riga, separandole con due punti (*:*).
- ✓ Commentare il codice utilizzando il simbolo ('), può seguire un'istruzione oppure occupare un'intera riga.

- ✓ Sistemi di numerazione: decimale (15), ottale (*&O*17), esadecimale (*&H*F).
- ✓ *ClipControls* = *False*, per avere maggiore velocità; è un processo che consente di determinare quali parti di un form saranno disegnate quando il form è visualizzato.
- ✓ *AutoRedraw* = *False*, per avere maggiore velocità; l'applicazione salva l'output grafico nella memoria per rivisualizzare la grafica nascosta da un'altra finestra.
- ✓ Utilizzare il metodo *Show* come prima istruzione nella routine *Form_Load*, permette di visualizzare il form mentre il compilatore esegue il codice.
- ✓ Per bitmap che rispondono solo a eventi clic usare il controllo immagine e non la casella immagine.
- ✓ Utilizzare bitmap compresse *.RLE (Run length Encoding).
- ✓ Registrare le immagini in un file di risorse ed usare la funzione *LoadResPicture* e non la proprietà *Picture* in fase di progettazione.
- ✓ *Set Picture1.Picture* = *Nothing* per svuotare la memoria.
- ✓ La proprietà *Image* di una casella immagine o di un form crea un bitmap con *AutoRedraw* = *True* anche se la proprietà *AutoRedraw* è impostata su False, per recuperare memoria:

```
pic.AutoRedraw = True
pic.Cls
pic.AutoRedraw = false
```

DISTRIBUZIONE DI APPLICAZIONI

Per poter utilizzare tutte le funzioni dell'Autocomposizione Installazione, è necessario salvare prima tutto il progetto e quindi chiuderlo.
Per creare il supporto di distribuzione eseguire i seguenti passi.

1. Autocomposizione Installazione: indicare il file di progetto;
2. includere un controllo dati o uno degli oggetti di accesso ai dati se si usa un database;
3. selezionare il drive di installazione;
4. selezionare i server OLE se devono essere distribuiti;
5. selezionare i file contenenti gli oggetti utilizzati nell'applicazione;
6. scegliere "Installa nella directory dell'applicazione";
7. selezionare i file necessari per il corretto funzionamento dell'applicazione.

IL DEBUGGER INTEGRATO

Maggiori sono le dimensioni delle applicazioni, maggiori sono le possibilità d'incorrere in errori, detti, in gergo, *bug* dal verbo debug (to) che significa mettere a punto, in pratica è un processo per riconoscere e correggere errori.
Dal progetto alla realizzazione possono essere variate sia le condizioni generali, sia le specifiche di base dell'applicazione.
Il cambiamento di queste ultime è una fonte inesauribile di errori, in altri casi gli errori sembrano presentarsi in modo casuale.
Per scovare questi errori, sarebbe molto comodo possedere uno strumento che permetta di scorrere il sorgente riga per riga, controllando, se possibile in una finestra attigua, il valore assunto dalle variabili da tenere sotto controllo.
In via generale la fase di debugging si articola nei seguenti quattro passi.

1. Verifica dell'esistenza di un errore. I casi macroscopici si hanno quando il sistema si "pianta", altre volte l'applicazione gira correttamente fino a quando non ha a che fare con un numero (o carattere) critico al quale è associato l'errore.
2. Ricerca della posizione dell'errore. Questo è più difficile quanto più lunga è l'applicazione, allora conviene sezionarla ed individuare quale parte contiene l'errore.
3. Ricerca della causa dell'errore.
4. Correzione dell'errore.

Tipi di errore

1. *Fatal*: immediata interruzione, sono catturabili.

2. *Error*: impediscono la generazione del .EXE, sono catturabili.
3. *Warning*: condizioni sospette, effetti non voluti genera in ogni caso il .EXE.
4. *Syntax error*: cursore nella posizione dell'errore, la riga dell'errore si colora di rosso e compare una finestra con il suggerimento.
5. *Compile time errors*: si verificano quando il codice non è corretto, attivare la casella *Strumenti/Opzioni/Ambiente/Controllo automatico sintassi.*
6. *Run time errors*: errori di semantica. Quando è rilevato un errore compare una finestra con il numero dell'errore ed il messaggio di errore. Facendo clic sul pulsante *Debug* è possibile passare alla finestra del codice, dove la riga che ha causato l'errore e racchiusa in un riquadro, correggere l'errore e quindi continuare l'esecuzione dell'applicazione con *Esegui/Continua*. Per sfruttare questo disattivare la casella *Strumenti/Opzioni/Avanzate/Intercettazione degli errori/Interrompi ad ogni errore.*
7. *Errori logici*: l'applicazione fornisce risultati diversi da quelli previsti (fa ciò che gli avete detto di fare, anziché ciò che volevate che facesse).

"Un'applicazione che gira, non è detto che giri bene; un test può essere usato per mostrare la presenza di errori, non per mostrare la loro assenza." Dijkstra.

Non si può sostenere che un'applicazione è buona solo perché ha superato dei test.
Un test può essere:

- ✓ *selettivo*: non garantisce la validità perché i dati d'input sono di natura soggettiva;
- ✓ *esaustivo*: impossibile assegnare tutti i valori ai dati d'input perché il tempo di verifica tenderebbe ad esplodere.

Quando un'applicazione comincia ad essere rattoppata al punto, di perdere la sua struttura è più semplice riscriverla. In genere sono a disposizione due tipi di debugger:

1. presenti in un menu nell'ambiente di sviluppo: **debugger integrati**;
2. applicazioni vere e proprie: **debugger stand alone**.

Le differenze tra i debugger, dal punto di vista delle potenzialità, sono numerose.
Per controllare un'applicazione semplice il debugger integrato è più che sufficiente, ma quando l'applicazione si fa più grande è sicuramente necessario utilizzare un debugger stand-alone.
È bene chiarire subito, per non generare troppi entusiasmi, che i debugger non trovano da soli gli errori, ma grazie al gran numero d'informazioni che è possibile ottenere tramite questi strumenti gli errori saltano fuori, quasi, da soli.
La programmazione sotto Windows aggiunge nuove difficoltà perché vi possono essere più applicazioni che utilizzano le stesse risorse.
UAE (*Unrecoverable Application Error*) significa che l'applicazione ha commesso un fatto grave e l'errore commesso non è controllabile: in questo caso l'applicazione è uccisa.
Quando il sorgente diventa molto lungo, è possibile marcare (**segnalibro**) uno o più punti del codice che devono essere raggiunti velocemente, l'istruzione sarà preceduta da un rettangolo azzurro ed è raggiungibile con segnalibro successivo/precedente.
Modifica/Segnalibri/Imposta/rimuovi segnalibro

Per eseguire il debug di un'applicazione, è necessario sapere qual è la modalità corrente, indicata nella barra del titolo:

- ✓ fase di progettazione;
- ✓ fase di esecuzione (*Esegui/Avvia F5*);
- ✓ modalità interruzione (*Esegui/Interrompi*).

Nella Finestra Debug è possibile controllare i valori di espressioni e variabili mentre sono eseguite le istruzioni nella fase di esecuzione; mentre in modalità interruzione si possono modificare i valori di variabili o proprietà. È costituita da due riquadri:

1. Controllo nella parte superiore, sono visualizzate le espressioni di cui si desidera controllare i valori durante l'esecuzione.
2. Immediata nella parte inferiore, sono visualizzate le informazioni ottenute eseguendo il debug delle istruzioni *Debug.Print [var][;]* nel codice o richieste digitando direttamente i

comandi nel riquadro con *Print (?*); al massimo è una calcolatrice *?10^2 = 100*. Questo metodo è migliore rispetto alle espressioni di controllo.

Trace

Per tracciare l'attività di un'applicazione si usavano, in mancanza di strumenti di debug, istruzioni *print,* le function *MessageBeep*, *MessageBox* e *wvsprintf* per controllare come precedeva l'applicazione.
L'unità base di esecuzione non è l'istruzione, ma la riga evidenziata dalla barra di esecuzione, scegliere *Debug/Esegui istruzione F8*.
Per evitare il tracing dentro le routine corrette (Tron e Troff del BASIC) scegliere *Debug/Esegui istruzione/routine MAIUSC+F8*.
È possibile uscire da una routine con *Esci da istruzione/routine MAIUSC+CTRL+F8*.
Per ignorare determinate sezioni di codice, ad esempio loop di grandi dimensioni scegliere *Debug/Esegui fino al cursore CTRL+F8.*
Debug/Imposta istruzione successiva CTRL+F9 solo se si trova nella stessa routine.
Solo in modalità interruzione *Esegui/Mostra istruzione successiva*.

Call dialog

È visualizzato un elenco delle chiamate alle routine attive nello stack solo in modalità interruzione, in altre parole le routine dell'applicazione che sono state avviate, ma non ancora completate. Scegliere *Visualizza*/*Stack di chiamate...CTRL+L*.

Watches

Visualizza/Finestra Immediata CTRL+G per eseguire le istruzioni digitate direttamente all'interno della finestra.
Visualizza/Finestra Variabili locali permette di verificare e/o modificare il valore assunto da tutti gli oggetti locali alla routine.
Visualizza/Finestra Espressioni di controllo permette di selezionare un insieme di variabili da monitorare durante l'esecuzione dell'applicazione.
Per aggiungere una variabile all'elenco di quelle visibili selezionare *Debug/Aggiungi espressione di controllo*

Breakpoints

In fase di esecuzione, un punto d'interruzione (arresto) indica che l'esecuzione deve essere interrotta immediatamente prima che sia eseguita una riga di codice specifica.
È possibile impostare o rimuovere un punto d'interruzione sia in modalità interruzione che in fase di progettazione (*Stop* del BASIC).
Per inserire un punto d'interruzione, posizionare il cursore sulla riga di codice nella quale si desidera interrompere l'esecuzione, scegliere *Debug/Imposta/Rimuovi punto di interruzione F9*
Debug/Rimuovi punti di interruzione CTRL+MAIUSC+F9
Quando si avvia l'applicazione, l'esecuzione continua fino a quando è raggiunto il punto d'interruzione.
Dopo che l'applicazione è stata interrotta, è possibile esaminare lo stato corrente dei dati con la voce *Controllo immediato MAIUSC+F9*, oppure posizionarsi sopra con il mouse e dopo alcuni istanti compare una tool tip con il valore.
Come scrivere applicazioni per il debugging:

- ✓ un'istruzione per riga;
- ✓ includere numerosi commenti;
- ✓ routine non più lunghe di venticinque righe;
- ✓ utilizzare il passaggio parametri;
- ✓ progettare routine LOOSELY COUPLED (lascamente connesse);
- ✓ test e debugging una sezione per volta.

Cattura e recupero degli errori

L'istruzione *On Error* attiva in un'applicazione, la gestione degli errori, in pratica un codice macchina in grado di catturare gli errori di esecuzione e, invece di bloccare l'applicazione, trasferisce l'elaborazione ad un'altra istruzione.

L'istruzione *Resume* permette di riprendere l'elaborazione dall'istruzione in cui si è verificato l'errore.

Prima di eseguire questa istruzione, il programmatore può individuare il tipo di errore e una sua descrizione, rispettivamente con le proprietà *Number* e *Description* dell'oggetto *Err*. Ogni errore è codificato con un numero intero, contenuto nella guida alal voce "Errori intercettabili".

```
Private Sub Command1_Click()
    On Error GoTo recupero
        Text2.Text = Text1.Text / 1936.27
        Exit Sub
recupero:
        MsgBox Err.Description & "Codice errore: " & Str(Err.Number), , "Errore run time"
        Text1.Text = 0
        Resume
End Sub
```

Dimensionare un componente

1. IDE
   1.1. Si sposta il mouse fino ad ottenere le dimensioni desiderate che appaiono sopra il puntatore del mouse se esso non è spostato per più di un secondo oppure sulla barra degli strumenti.
   1.2. Finestra proprietà *Height* e *Width*.
2. Codice: *If Text1.Width < 567 Then Text1.Width = 567*

Per ridimensionare la Form si usa la routine *Private Sub Form_Resize()*

Posizionare un componente

1. IDE
   1.1. Si trascina il mouse fino ad ottenere la posizione desiderata visualizzata sulla barra degli strumenti.
   1.2. Finestra proprietà *Top* e *Left*.
2. Codice: *Text1.Left = 120: Text1.Top = 120*

Sovrapposizione di componenti

1. IDE clic con il pulsante destro del mouse Porta in primo (secondo) piano.
2. Codice: *Command1.ZOrder (0)* è in primo piano, 1 secondo piano.

Tipo di font

1. Finestra proprietà *Font*
2. Codice

```
Text1.FontName = "Arial"        'Tipo di carattere
Text1.FontBold = True           'Stile: Grassetto
Text1.FontItalic = True         'Stile: Corsivo
Text1.FontSize = 14             'Punti: 14
Text1.FontUnderline = True      'Effetti: Sottolineato
Text1.FontStrikethru = True     'Effetti: Barrato
```

È possibile ottenere un elenco dei font installati nel sistema.

```
For i = 0 To Screen.FontCount - 1
    List1.AddItem (Screen.Fonts(i))
Next i
Text1.FontName = Screen.Fonts(List1.ListIndex)
```

Centrare o spostare un oggetto sui bordi rispetto ad un altro oggetto

```
Public Function Centrare(o As Object)
    o.Left = (o.Parent.Width - o.Width) / 2
    o.Top = (o.Parent.Height - o.Height) / 2
End Function
Public Function AllineaDestra(o As Object)
    o.Left = o.Parent.Width - o.Width
End Function
Public Function AllineaBasso(o As Object)
    o.Top = o.Parent.Height - o.Height
End Function
Public Function AllineaSinistra(o As Object)
    o.Left = 1
End Function
Public Function AllineaAlto(o As Object)
    o.Top = 1
End Function
Call Centrare(Text1)
```

Purtroppo l'oggetto Form non ha la proprietà Parent, si deve scrivere

```
Public Function CentrareForm(f As Form)
```

```
        f.Left = (Screen.Width - f.Width) / 2
        f.Top = (Screen.Height - f.Height) / 2
End Function
Public Function FormAllineaDestra(f As Object)
        f.Left = Screen.Width - f.Width
End Function
Public Function FormAllineaBasso(f As Object)
        f.Top = Screen.Height - f.Height
End Function
Public Function FormAllineaSinistra(f As Object)
        f.Left = 1
End Function
Call CentrareForm(Form1)
```

AllineAlto e AllineaSinistra sono uguali.
Ridimensiona altezza e ampiezza dell'oggetto nella form.

```
Private Sub Form_Resize()
        Text1.Width = Form1.Width
        Text1.Height = Form1.Height
End Sub
```

Per impedire all'utente l'uso del pulsante di chiusura di una Form

```
Private Sub Form_Unload(Cancel As Integer)
        Cancel = -1
End Sub
```

Focus

Indica il componente attivo, in pratica dove è posizionato il cursore. Il TAB, gli shortcut di tastiera o un clic del mouse su un altro oggetto spostano il focus.

1. IDE proprietà TabIndex, o in base alla sequenza di creazione.
2. Codice:

```
Text1.SetFocus
Label2.Caption = "&Progetto"
Label2.TabIndex = 0
Text2.TabIndex = 1          'premendo ALT+P il focus va su Text2
Text3.TabStop = False       'non uso TAB per accedere a Text3
```

Verifica della correttezza delle informazioni digitate dall'utente

Quando l'utente si sposta su un altro controllo che ha la proprietà CausesValidation attiva (True, default), è generato l'evento Validate, nel quale il programmatore può effettuare tutti i controlli necessari sui dati inseriti per autorizzare o negare lo spostamento del focus con il valore del parametro Cancel. Nell'esempio per spostare il focus, Text1 deve avere una lunghezza di almeno tre caratteri.

```
Public Sub Text1_Validate(Cancel As Boolean)
        If Len(Text1) < 3 Then
                Cancel = True
        End If
End Sub
```

Agire prima che il componente riceva il focus

Quando il focus raggiunge un oggetto, è generato l'evento GotFocus. Se a tale oggetto è associata una routine, essa è eseguita prima che l'oggetto prenda il controllo. Nell'esempio si seleziona il testo in Text1 prima che Text1 prenda il focus.

```
Private Sub Text1_GotFocus()
        Text1.SelStart = 0
        Text1.SelLength = Len(Text1)
End Sub
```

Agire quando il focus lascia il componente

L'evento LostFocus è generato quando un oggetto perde il focus. È utile per verificare ciò

che ha fatto l'utente. Nell'esempio si tolgono gli spazi iniziali e finali superflui.

```
Private Sub Text1_LostFocus()
        Text1 = Trim(Text1)
End Sub
```

Come aprire più form dello stesso tipo

Se l'applicazione deve aprire più copie dello stessa form per visualizzare informazioni diverse, è necessario un sistema che permetta ad ogni singola copia della form di individuare le proprie informazioni: basta creare una o più variabili Public all'interno della seconda form, esse riceveranno dalla form principale i valori. Ci sono, poi, due possibili modi di chiudere la form principale.

1. Le copie delle form create dall prima sono dipendenti (Show, Me) e la chiusura di quest'ultima genera la loro chiusura.
2. Le copie delle form create dalla prima sono indipendenti (Show) e la chiusura di quest'ultima non genera la loro chiusura, ma restano aperte.

L'applicazione fa riferimento alla form attraverso il suo nome oppure la parola Me.

```
Private Sub CreaNuovaForm()
        Dim NuovaForm As Form2
        Set NuovaForm = New Form2
        NuovaForm.Dati = Time          'variabile in form2 che contiene l'ora
        NuovaForm.Show, Me
End Sub
Private Sub Command1_Click()
        Call CreaNuovaForm             'form1 crea tante copie di form2
End Sub
Public Dati As String                  'dichiarazione in form2
Private Sub Command1_Click()
        Unload Me
End Sub
Private Sub Form_Load()
        Label1 = Hour(Dati)
        Label2 = Minute(Dati)
        Label3 = Second(Dati)
End Sub
```

La form può essere.

- ✓ Non Modal (Show 0) l'utente può usare altre form, mentre form1 è visualizzata.
- ✓ Modal (Show 1) tutte le altre form dell'applicazione sono inattive

CommandButton

1. IDE Finestra proprietà *Default* = *True*
2. Codice: *Command1.Default = True*

All'interno di una Form c'è un solo pulsante di default per il quale è generato l'evento Click, anche dal tasto INVIO. Esistono due stili di pulsanti.

1. Testuali IDE Finestra proprietà Style = *Standard*
2. Grafici IDE Finestra proprietà *Style* = *Graphical*

Si deve indicare l'icona con il pathname.

1. IDE Finestra proprietà *Picture* indica l'icona che appare sul pulsante. Se il pulsante è stato disattivato con *Enabled* = *False* è visualizzata una versione grigia dell'icona.
2. Codice: C*ommand1.Picture = LoadPicture ("c:\...")*

IDE Finestra proprietà *DownPicture* indica l'icona che deve essere visualizzata mentre il pulsante è premuto.

IDE Finestra proprietà *DisablePicture* indica l'icona che deve essere visualizzata se il pulsante è stato disattivato con *Enabled* = *False*.

```
Private Sub Command1_Click()
```

```
    Dim Risposta As VbMsgBoxResult
    Risposta = MsgBox("Dati modificati: salvarli?", vbYesNoCancel, "AVVISO")
    Select Case Risposta
        Case vbYes: '...
        Case vbNo: '...
        Case vbCancel: '...
    End Select
    Unload Me
End Sub
```

TextBox

1. IDE Finestra proprietà *Text = Ciao Mondo*
2. Codice:

```
Private Sub Command1_Click()
        Text1.Text = "Ciao Mondo"
        'inserire la stringa all'inizio
        Text1.Text = "Prima di " & Text1.Text
        'inserire la stringa alla fine
        Text1.Text = Text1.Text & " Dopo"
        'inserire la stringa all'interno
        Text1.SelStart = 5          'definisce il punto d'inserimento
        Text1.SelText = "Nuovo "    'inserisce la nuova stringa
    End Sub
```

Se la posizione indicata con SelStart (il primo carattere è indicato da zero) supera la lunghezza del testo, non ci sarà errore, la nuova stringa è accodata.
La proprietà MaxLength definisce il numero massimo di caratteri che la TextBox accetta, se vale 0, l'utente può inserire 32.000 caratteri.

```
Private Sub Command1_Click()
    If (Text1.MaxLength = 0) Then Text1.MaxLength = 5
End Sub
```

Se il testo è lungo si usano: IDE Finestra proprietà *MultiLine = True e ScrollBar = 1,2,3*
È possibile estrarre la parte di stringa che il programmatore ha selezionato.

```
Private Sub Command1_Click()
    Dim s As String
    Text1.Text = "Ciao Mondo"
    Text1.SelStart = 5
    Text1.SelLength = 5             'seleziono il numero di caratteri
    Text2.Text = Text1.SelText
    If (Text1.SelLength > 0) Then   'la if esegue la stessa operazione
        s = Mid$(Text1.Text, Text1.SelStart + 1, Text1.SelLength)
    End If
    Text2.Text = s
    Text1.SelStart = 0
    Text1.SelLength = Len(Text1.Text)   'seleziono tutto il testo
End Sub
Private Sub Command1_Click()
    Dim s As String
    Text1.Text = "Ciao Mondo"
    Text1.SelStart = 5
    Text1.SelLength = 5
    Text1.SelText = ""              'elimina il testo selezionato
    If (Text1.SelLength > 0) Then   'la if esegue la stessa operazione
    s=Left$(Text1.Text,Text1.SelStart)&Right$(Text1.Text,Len(Text1)-
    (Text1.SelStart+Text1.SelLength))
    End If
```

```
        Text2.Text = s
End Sub
Private Sub Command1_Click()
        Text1.Text = "Ciao Mondo"
        'Text1.SelStart = Len(Text1.Text)
        Text1.SelStart = 5
        Text1.SetFocus                  'posiziona il cursore nella posizione desiderata
End Sub
```

IDE Finestra proprietà: *DataFormat* è possibile definire in modo guidato (come Excel) il formato con cui i dati sono mostrati all'utente.
IDE Finestra proprietà *BackColor ForeColor* impostano il colore di sfondo ed il colore del testo. Per i colori specifici (Tavolozza), per quelli pre impostati (Sistema).

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
        'impedisce alla TextBox di emettere un beep se è stato premuto il tasto INVIO (13)
        If KeyAscii = 13 Then KeyAscii = 0 'allora considera come nessun tasto premuto
End Sub
```

Label

La dimensione può essere dinamica e variare in relazione alla quantità e alle dimensioni del testo da visualizzare.
IDE Finestra proprietà *Autosize = True*

ListBox e ComboBox

- ✓ Ogni elemento che appare nella ListBox può essere inserito, cancellato o modificato in fase di progettazione (IDE) oppure in fase di esecuzione.
- ✓ Può visualizzare gli elementi in base all'ordine alfabetico.
- ✓ Ad ogni elemento può essere associato un valore.
- ✓ Per l'elemento selezionato può essere acquisito sia l'elemento sia il valore associato.
- ✓ Si può abilitare l'utente ad effettuare la selezione di più elementi ed è possibile acquisirli in sequenza anche con il relativo valore associato, con la proprietà Multiselect (None seleziona un solo elemento, Simple seleziona più elementi contigui, Extended seleziona più elementi anche non contigui).
- ✓ Per elenchi lunghi o i cui elementi possono essere variati dall'utente è possibile salvare e/o recuperare gli elementi sul disco.

1. IDE Finestra proprietà *List* ogni valore sarà su righe separate da CTRL + INVIO
2. Codice:

```
Private Sub Command1_Click()
        Dim a As String
        a = ("Orario: " & Time)     'parametro di AddItem valore da inserire
        List1.AddItem a, 0          'indica la posizione d'inserimento
End Sub
```

IDE Finestra proprietà *Sorted* ordina i dati (non bisogna però indicare la posizione d'inserimento), non è modificabile a livello codice (runtime).

```
Private Sub Command1_Click()
        Dim a As String
        Dim i As Integer
        Open "C:\esempi\output\dati.txt" For Input As #1
        While Not EOF(1)
                Input #1, a
                List1.AddItem a             'inserire elementi caricandoli da un file
        Wend
        Close #1
        Open "C:\esempi\output\dati.txt" For Append As #1
        For i = 0 To List1.ListCount – 1    'individua il numero di elementi
                Print #1, List1.List(i)     'copia su file degli elementi
```

```
        Next i
        Close #1
End Sub
Private Sub Command2_Click()
        Dim i As Integer
        List1.List(0) = "Carlo Bianchi"          'modificare un elemento
        i = List1.ListIndex                      ‘individua l’elemento selezionato
        List1.RemoveItem (i)                     'elimina un elemento selezionato
        List1.Clear                              'elimina tutti gli elementi
End Sub
```

ListIndex = –1 se nessun elemento è stato selezionato, altrimenti l’indice a partire da 0.

Assegnare un valore ad un elemento

Ogni elemento è associato al testo descrittivo che appare dentro la ListBox, ma è anche possibile associare un valore numerico di tipo Long in fase di progettazione (IDE) con la proprietà *ItemData* oppure in fase di esecuzione. L’associazione fra il testo ed il valore avviene per posizione: il primo valore è associato al primo elemento.

```
Private Sub Command2_Click()
        Dim i As Integer
        List1.ItemData(2) = 123                  'assegna un valore al terzo elemento
        List1.ItemData(List1.ListIndex) = 124    'assegna un valore all'elemento selezionato
        For i = 0 To List1.ListCount – 1    'assegna il valore 0 a tutti gli elementi selezionati
                If List1.Selected(i) Then List1.ItemData(i) = 0
        Next i
        Text1.Text = Str(List1.ItemData(2))      'acquisisce il valore associato all'elemento
        'acquisisce il valore associato all'elemento selezionato
        Text1.Text = Str(List1.ItemData(List1.ListIndex))
End Sub
```

Individurae l’indice dell’elemento più recentemente inserito

```
Private Sub Command1_Click()
        If List1.NewIndex <> -1 Then             ‘-1 se la ListBox è vuota
                MsgBox "L'indice dell'elemento inserito per ultimo è " & (List1.NewIndex + 1)
        End If
End Sub
```

Estrarre l’elemento selezionato

```
Private Sub List1_Click()
        Dim i As Long
        i = List1.ListIndex
        Text1.Text = List1.List(i)
End Sub
```

Se la proprietà Multiselect è attiva, per estrarre gli elementi selezionati.

```
Private Sub Command1_Click()
        Dim s As String
        Dim i As Integer
        For i = 0 To List1.ListCount - 1
                If List1.Selected(i) = True Then
                        'acquisisce l'elemento con indice i
                        s = List1.List(i)
                        'memorizza la stringa estratta in un'altra lista
                        List2.AddItem (s)
                End If
        Next i
End Sub
```

CheckBox
Quadratini bianchi che disegnano un segno di spunta al loro interno in seguito ad una selezione del mouse. Alla pressione del Command, si presuppone che l'utente abbia terminato di compilare il form, nella TextBox compariranno i CheckBox spuntati.

```
Private Sub Command1_Click()
    If Check1.Value = Checked Then
        Text1.Text = Text1.Text & "CheckBox1" & Chr(13) + Chr(10)
    End If
    If Check2.Value = Checked Then
        Text1.Text = Text1.Text & "CheckBox2" & Chr(13) + Chr(10)
    End If
    If Check3.Value = Checked Then
        Text1.Text = Text1.Text & "CheckBox3" & Chr(13) + Chr(10)
    End If
    If Check4.Value = Checked Then
        Text1.Text = Text1.Text & "CheckBox4" & Chr(13) + Chr(10)
    End If
End Sub
```

OptionButton
1. IDE Finestra proprietà *Value = True*
2. Codice: *Option1.Value = True Text1.Text = Option1.Value*

Permette di effettuare una scelta fra più possibilità. Abitualmente sono inseriti in un Frame perché VB provvede automaticamente a deselezionare gli altri quando l'utente seleziona un OptionButton. L'evento Click è generato solo quando l'utente esegue un click ed esso assume valore True, quindi nell'esempio il ramo else non è mai eseguito.

```
Private Sub Option1_Click()
    If (Option1.Value = True) Then
        Option1.Caption = "Sono selezionato"
    Else
        Option1.Caption = "Sono deselezionato"
    End If
End Sub
```

ScrollBar
Usate all'interno di altri controlli, sono di due tipi *HScrollBar* (Orizzontale) *VScrollBar* (verticale). Clic sulla ScrollBar genera l'evento *Click*, clic sul cursore genera l'evento *Scroll*. Per acquisire la posizione del cursore si controlla la proprietà *Value*, restituisce un valore tra *Min* e *Max*, default 0-32767.

```
Private Sub sposta(i As Integer)
    List1.AddItem (i)
End Sub
Private Sub HScroll1_Change()
    sposta (HScroll1.Value)
End Sub
Private Sub HScroll1_Scroll()
    sposta (HScroll1.Value)
End Sub
```

Timer
Permette di generare automaticamente un evento al trascorrere di un periodo di tempo espresso in milli secondi, significa che, se si volesse far apparire una MessageBox ogni secondo, bisognerebbe impostare 1.000. Non è garantito che l'evento sia generato allo scadere del tempo impostato, anzi se il PC è carico di lavoro, l'evento è elaborato in

ritardo. La proprietà *Interval* è usata per definire il numero di milli secondi, il tempo massimo è uguale a 1 minuto, 5 secondi e 535 milli secondi (65.535). L'attivazione del Timer è automatica all'avvio dell'applicazione, a meno che la proprietà *Timer1*.*Enabled* = *False*. La sua riattivazione causa l'inizio del conto alla rovescia del periodo, perciò non è necessario prevedere la sua riattivazione nella routine di gestione dell'evento.

```
Private Sub Timer1_Timer()
    List1.AddItem (Time)
End Sub
```

Per gestire tempi superiori, impostare *Interval* = 60000 e usare una variabile statica che è incrementata di un'unità dall'evento Timer, la variabile indica i minuti trascorsi.

```
Private Sub Timer1_Timer()
    Static minuti As Integer
    If minuti = 60 Then
        MsgBox "E' trascorsa un'ora!!"
        minuti = 1
    Else
        minuti = minuti + 1
    End If
End Sub
```

CommonDialog (Microsoft Common Dialog Control 6.0)

Per conoscere il nome del file che l'utente ha selezionato: *Filename* (nome del file completo di drive, cartella ed estensione), *FileTitle* (solo nome ed estensione).
Volendo conoscere il tipo di file selezionato dall'utente: *FilterIndex*.

```
Private Sub Command1_Click()
    CommonDialog1.DialogTitle = "Apri"          'titolo della finestra
    CommonDialog1.InitDir = "C:\"               'directory iniziale
    CommonDialog1.DefaultExt = ".txt"
    CommonDialog1.Filter = "Documenti di testo|*.txt" & "|" & "Tutti i file (*.*)"
    CommonDialog1.Flags = cdlOFNHelpButton
    CommonDialog1.ShowOpen
    If CommonDialog1.FileName <> "" Then
        MsgBox CommonDialog1.FileName
    End If
End Sub
```

La if verifica se la finestra è stata chiusa con OK dopo che l'utente ha selezionato il file da aprire, allora si richiama una funzione che ha come parametro il nome del file.

```
Private Sub Command2_Click()
    CommonDialog1.DialogTitle = "Salva con nome"
    CommonDialog1.InitDir = "C:\"
    CommonDialog1.DefaultExt = ".txt"
    CommonDialog1.Filter = "Documenti di testo|*.txt" & "|" & "Tutti i file (*.*)"
    CommonDialog1.Flags = cdlOFNHelpButton
    CommonDialog1.Flags = cdlOFNOverwritePrompt
    CommonDialog1.Flags = cdlOFNHideReadOnly
    CommonDialog1.ShowSave
End Sub
Private Sub Command3_Click()
    CommonDialog1.DialogTitle = "Stampa"
    CommonDialog1.Flags = cdlOFNHelpButton
    CommonDialog1.Flags = cdlPDPrintSetup      'imposta stampante
    CommonDialog1.ShowPrinter
End Sub
Private Sub Command4_Click()
```

```
    CommonDialog1.ShowColor                 'selezione dei colori dalla tavolozza
End Sub
Private Sub Command5_Click()
    CommonDialog1.Flags = cdlCFBoth
    CommonDialog1.Flags = cdlCFPrinterFonts
    CommonDialog1.Flags = cdlCFScreenFonts
    CommonDialog1.ShowFont                  'selezione dei font o caratteri
End Sub
Private Sub Command6_Click()
    CommonDialog1.HelpFile = "c:\esempi\scuola\automi\automi.hlp"
    CommonDialog1.HelpCommand = cdlHelpContents
    CommonDialog1.ShowHelp
End Sub
```

ImageList (Microsoft Windows Common Controls 6.0)
Non è visibile in run time, quindi deve essere disegnato in fase di progettazione su un form. È un contenitore d'immagini (BMP, ICO, CUR, JPG, GIF) al quale altri controlli possono accedere: in fase di progettazione sono caricate nel controllo le immagini evitando così il rallentamento dell'applicazione in esecuzione. È inoltre possibile aggiungere immagini alla lista anche a run time. Ha una collezione d'immagini di nome *ListImages*; ogni immagine costituisce un oggetto, è possibile riferirsi a ciascun oggetto mediante un indice numerico (*Index*) o una chiave alfanumerica (*Key*) e un valore (*Tag*) per descrivere l'oggetto.
IDE Finestra proprietà *(personalizzate) Generale Immagini*
Aggiunta d'immagini: metodo *Add*.
Rimozione d'immagini: metodo *Remove*.
Rimozione di tutte le immagini: metodo *Clear*.
Visualizza un'immagine su un form: *PaintPicture ImageList1.ListImages(1).Picture, 0, 0*
E in una PictureBox: *ImageList1.ListImages(1).Draw Picture1.hDC, 0, 0*

ImageCombo (Microsoft Windows Common Controls 6.0)
È una ComboBox che utilizza immagini.
Proprietà ImageList: riferimento al controllo ImageList che contiene le immagini da utilizzare.

Strumenti/Editor di menu... CTRL + E
1. Nella casella *Caption* digitare la voce di menu, inserendo & davanti al carattere che sarà usato per la selezione da tastiera con ALT.
2. Nella casella *Name* digitare l'identificatore del menu, quando l'utente seleziona la voce genera un evento la cui routine ha il nome composto dall'identificatore e il suffisso _Click.
3. Se la voce di menu è un sotto menu, allora fare clic sulla freccia destra, appare preceduta da una serie di punti.
4. Premere INVIO oppure fare clic su *Successivo.*
5. Ripetere i passi precedenti per ogni voce di menu.

Per disabiltare una voce di menu si agisce sulla proprietà *Enable*, essa rimarrà visibile ma non selezionabile. Agendo sulla proprietà *Visible* una voce di menu sparisce senza lasciare spazi. Le voci di menu possono essere precedute dal segno di spunta agendo sulla proprietà *Chcked*.
Per selezionare dei tasti di scelta rapida, selezionare la voce di menu e quindi la voce *ShortCut*.
Per raggruppare le voci di menu tra loro mediante una linea di separazione, occorre creare un menu fittizio che abbia come campo Caption il simbolo "-".

Toolbar (Microsoft Windows Common Controls 6.0)
Crea una barra degli strumenti: è un'area rettangolare che occupa la parte superiore del form. È possibile averne più di una e allineate su bordi diversi con la proprietà *Align*.
IDE Finestra proprietà *(personalizzate) Pulsanti Inserisci pulsante*, per creare i tooltip si deve digitare il testo nella casella *ToolTipText*.
L'inserimento dell'icona all'interno dei pulsanti avviene nel modo seguente.
1. Inserire le icone nell'*ImageList* (non possono essere caricate nella Toolbar).
2. IDE Finestra proprietà *(personalizzate) Generale ImageList* collega i due componenti.
3. IDE Finestra proprietà *(personalizzate) Pulsanti Image* associa le icone ai pulsanti.
4. IDE Finestra proprietà *(personalizzate) Pulsanti Inserisci pulsante* ripetere le operazioni per ogni pulsante.

Quando l'utente fa clic su un pulsante, genera l'evento *ButtonClick*; per gestirlo deve ricorrere al parametro che questa funzione riceve ed interrogarlo con le proprietà:
- ✓ *Index* restituisce un numero intero attribuito dall'IDE in creazione;
- ✓ *Key* restituisce una stringa attribuita dall'IDE in creazione.

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
    Select Case Button.Index                    'Button.Key
        Case 1                                  'Key
            MsgBox "Uno"
        Case 2                                  'Key
            MsgBox "Due"
    End Select
End Sub
```

Per gestire dall'applicazione la Toolbar: *Toolbar1.Buttons.Item(1).Enabled = False*

Normalmente le toolbar sono impostate in fase di progettazione mediante una procedura guidata. Il wizard è un add-in che deve essere installato con *Aggiunte/Gestione aggiunte/Creazione guidata VB6.*

StatusBar (Microsoft Windows Common Controls 6.0)
La barra di stato è visualizzata nella parte inferiore della finestra dell'applicazione, ha una collezione di oggetti Panel, ciascun oggetto è una parte della barra che può contenere informazioni.
1. IDE Finestra proprietà *Style = sbrNormal*
2. Codice: *StatusBar1.Style = sbrSimple*

Creare un pannello all'interno della StatusBar e modificare le dimensioni
IDE Finestra proprietà *(personalizzate) Pannelli Inserisci pannello,* inserito a destra
1. IDE Finestra proprietà *(personalizzate) Pannelli Larghezza minima*
2. Codice: *StatusBar1.Panels.Item(2).MinWidth = 1000*
1. IDE Finestra proprietà *(personalizzate) Pannelli a*spetto del pannello *Bevel*
2. Codice: *SSTab1. Panels.Item(2).Bevel = sbrRaised*

Visualizzare lo stato della tastiera
1. IDE Finestra proprietà *(personalizzate) Pannelli Style*
2. Codice: *StatusBar1.Panels.Item(1).Style = sbrCaps*

Inserire un'immagine o testo
1. IDE Finestra proprietà *(personalizzate) Pannelli Immagine*
2. Codice: *StatusBar1.Panels.Item(2).Picture (.Text)= LoadPicture (c:\..)*

Per inserire testo nella Statusbar definita *sbrSimple*, si usa la proprietà *SimpleText*.

TreeView (Microsoft Windows Common Controls 6.0)
Visualizza una serie di elementi sotto forma di una struttura gerarchica: con oggetti genitori (parent) che possono avere figli (child). Ha una collezione di nodi (nodes) ai quali sono associate diverse proprietà.
La proprietà *LineStyle* definisce se le linee che uniscono fra loro i nodi devono, oppure no

(0), mostrare i sottonodi (1). La proprietà *Style* determina l'aspetto del controllo.
Gli oggetti Node possono essere aggiunti solo in run time con il metodo Add.

```
Dim n As Node
Set n = TreeView1.Nodes.Add(, , "Chiave1", "Primo", 1, 4)
Set n = TreeView1.Nodes.Add("Chiave1", tvwChild, "Chiave11", "Ciao")
Set n = TreeView1.Nodes.Add(, , "Chiave2", "Secondo", 2, 4)
Set n = TreeView1.Nodes.Add(, , "Chiave3", "Terzo", 3, 4)
```

ListView (Microsoft Windows Common Controls 6.0)
Permette di visualizzare un elenco di oggetti sotto forma di icone (*Icon*), icone piccole (*SmallIcon*), elenco (*List*), dettagli (*Report*).
Gli oggetti possono essere aggiunti solo in run time con il metodo Add.

ProgressBar (Microsoft Windows Common Controls 6.0)
Indica lo stato di avanzamento di un lavoro espresso graficamente in percentuale, 0 valore minimo non appare la barra, 50 sino a metà, 100 valore massimo interamente. Il valore dell'incremento (sempre compreso Min e Max) è dato da 100 diviso il numero di operazioni da eseguire.

```
Private Sub esegue(x As Integer)
        Dim i As Integer
        For i = 1 To x
                'call esegui qualcosa....
                ProgressBar1.Value = ProgressBar1.Value + (100 / x)
        Next i
End Sub
```

Esempio con un timer: al primo evento timer è assegnato il valore Min della barra alla variabile v, che a sua volta è poi assegnata alla proprietà Value della barra. Negli istanti successivi Value cambia, fino al valore Max.

```
Private Sub Timer1_Timer()
   Static v As Integer
   If (v = 0) Then v = ProgressBar1.Min
   ProgressBar1.Value = v
   v = v + 1
   If (v >= ProgressBar1.Max) Then Timer1.Enabled = False
End Sub
```

Slider (Microsoft Windows Common Controls 6.0)
Sistema di selezione di un valore compreso tra due limiti specificabili. Le tacche si chiamano Tick ed il numero frequenza. Per impostare questo valore si usa una frazione, se lo Slider va da 0 a 100 e il Tick deve rappresentare 1/20, allora si fa Max/20 = 5, è utile quando i limiti sono dinamici.

```
Slider1.Min = 0                          'valore minimo -32768
Slider1.Max = 100                        'valore massimo +32767, default 0-32767
Slider1.TickFrequency = Slider1.Max / 20 'frequenza dei Tick sullo Slider
Slider1.LargeChange = 10                 'incremento di PagSu e pagGiù o un clic
Slider1.SmallChange = 1                  'incremento dei tasti cursore
Slider1.Orientation = 1                  '0= orizzontale 1 = verticale
Slider1.TickStyle = 3                    'ci sono 4 stili grafici
MsgBox "Valore minimo " & Slider1.Min
MsgBox "Numero di Tick sullo Slider " & Slider1.GetNumTicks      'TickFrequency + 1
```

La selezione di un intervallo di valori avviene trascinando lo Slider e tenendo premuto contemporaneamente MAIUSC.

```
Dim s As Integer
Private Sub Slider1_MouseDown(Button As Integer, Shift As Integer, x As Single, y As
```

```
Single)
   If Shift = vbShiftMask Then
      Slider1.SelectRange = True
      Slider1.SelLength = 0
      s = Slider1.Value
   Else
      Slider1.SelectRange = False
   End If
End Sub
Private Sub Slider1_Scroll()
    'È possibile visualizzare il valore selezionato come tooltip.
   Slider1.Text = "Valore = " & Slider1.Value
   With Slider1
      If .Value > s Then
         .SelStart = s
         .SelLength = .Value - s
      Else
         .SelStart = .Value
         .SelLength = s - .Value
      End If
   End With
End Sub
Private Sub Slider1_MouseUp(Button As Integer, Shift As Integer, x As Single, y As
Single)
   If Slider1.SelectRange Then
      MsgBox "Inizio a " & Str(s) & " , fine a " & Slider1.Value & ", intervallo pari a  " &
Slider1.SelLength
   End If
End Sub
```

TabStrip (Microsoft Windows Common Controls 6.0)

Permette di visualizzare ed utilizzare una finestra di dialogo a schede; esiste un altro controllo SSTab, che gestisce lo stesso tipo di finestra.

IDE Finestra proprietà *(personalizzate) Generale Schede*

Per posizionare dei controlli sulle singole schede, siccome TabStrip non è un controllo contenitore (container), bisogna creare una matrice di controlli contenitori (per esempio, PictureBox o Frame) nei quali si disegnano tutti i controlli che servono.

```
Option Explicit
'mioframe è una matrice di Frame
'i frame sono spostati e adattati a TabStrip e resi non visibili
Private Sub Form_Load()
   Dim fr As Frame
   For Each fr In mioframe
        fr.Move TabStrip1.ClientLeft, TabStrip1.ClientTop, TabStrip1.ClientWidth,
        TabStrip1.ClientHeight
      fr.BorderStyle = 0
      fr.Visible = False
   Next
End Sub
'quando si fa clic sulla linguetta di una scheda, il frame corrispondente
'deve essere reso visibile e tutti gli altri nascosti
Private Sub TabStrip1_Click()
   Dim fr As Frame
   For Each fr In mioframe
```

```
      If (fr.Index = TabStrip1.SelectedItem.Index - 1) Then
         fr.Visible = True
      Else
         fr.Visible = False
      End If
   Next
End Sub
'facendo clic su una scheda bisogna convalidare o analizzare cosa l'utente ha inserito
'sulla scheda che si sta per abbandonare, l'evento BeforeClick permette questo
'Su ogni frame c'è una TextBox, non si passa alla scheda cliccata se le
'TextBox sono vuote
Private Sub TabStrip1_BeforeClick(Cancel As Integer)
   Select Case TabStrip1.SelectedItem.Index
      Case 1
         'prima scheda, se Text1 è vuota non cambia scheda
         If Text1.Text = "" Then Cancel = True
      Case 2
         'seconda scheda, se Text2 è vuota non cambia scheda
         If Text2.Text = "" Then Cancel = True
   End Select
End Sub
```

SSTab (Microsoft Tabbed Dialog Control 6.0)

Permette di aggiungere al progetto una finestra di dialogo a schede, accessibili una alla volta. Il controllo fornisce un gruppo di schede, ognuna delle quali può essere contenitore per altri controlli. I Tab possono apparire su uno dei quattro lati impostando la proprietà: *TabOrietation*.

```
Private Sub Command1_Click()
        Dim n As Integer
        n = SSTab1.Tab
        MsgBox "La scheda attiva è " & n                ‘acquisire la scheda attiva
        MsgBox "Schede presenti " & SSTab1.Tabs    ‘acquisisce il numero di schede
        SSTab1.Tab = 0                                  ‘quale scheda deve essere attiva
End Sub
```

TabMaxWidth larghezza massima della scheda, con il valore 0 (predefinito), le dimensioni variano con il testo che devono contenere. *TabsPerRow* definisce quanti Tab per riga.

Definire il testo del Tab

1. IDE Finestra proprietà *(personalizzate) Generale TabCaption*
2. Codice: *SSTab1.TabCaption(1) = "Prova"*

Inserire un’icona all’inizio del testo del Tab

1. IDE Finestra proprietà *(personalizzate) Immagine*
2. Codice: *SSTab1.TabPicture(1) = LoadPicture (“C:\...)*

Disabilitare e abilitare una scheda

1. IDE Finestra proprietà *Enabled = True*
2. Codice: *SSTab1.TabEnabled(1) = False* *‘agisce sulla singola scheda*

```
Public Function DisabilitaTab(x As Integer)
        SSTab1.TabEnabled(x) = False                    'disabilita la scheda
        If x > 0 Then                                   'se la scheda non è la prima
                SSTab1.Tab = (x - 1)                    'allora seleziona la precedente
        Else                                            ‘altrimenti ci sono più schede
                If SSTab1.Tabs > 1 Then SSTab1.Tab = SSTab1.Tabs - 1
        End If
End Function
Private Sub Command1_Click()
```

DisabilitaTab (3)
End Sub

Calendar (Microsoft Calendar Control 8.0)
Si agisce sulla proprietà *Day* o *Month* per personalizzare il calendario.
- ✓ La data che il calendario deve considerare di riferimento.
- ✓ Il giorno della settimana che il calendario deve considerare come riferimento.
- ✓ Il formato del giorno: breve, abbreviata, estesa.
- ✓ Il formato del mese: breve, estesa.
- ✓ La visualizzazione: incassato, in rilievo, piatto.

I due eventi *NewMonth* e *NewYear* permettono all'utente di spostarsi su mesi o anni diversi da quello corrente.
Acquisire la data selezionata sul calendario
Private Sub Calendar1_Click()
Text1.Text = Calendar1
End Sub
Eseguita in risposta ad un clic su Calendar ad eccezione delle ComboBox di mese e anno.
Private Sub Calendar1_AfterUpdate()
Eseguita in risposta ad un'azione dell'utente che cerca di spostare il giorno selezionato e dopo che tale operazione è stata eseguita, sempre che Cancel non sia 1.
Impedire lo spostamento del giorno
È possibile analizzare il giorno selezionato dall'utente prima che lo spostamento diventi operativo ed impedire questa operazione agendo su Cancel.
Private Sub Calendar1_BeforeUpdate(Cancel As Integer)
'controlla se si tratta di una domenica
If Weekday(Calendar1) = vbSunday Then 'segnale all'utente
MsgBox "Non è possibile selezionare la domenica"
Cancel = 1 'impedisce lo spostamento
End If
End Sub
Definire la data di riferimento
È la data che il calendario deve mostrare, si usa la proprietà *Value* con valori di tipo Date.
Calendar1.Value = #1/10/2000# '10 gennaio 2000!

MSComm (Microsoft Com Control 6.0)
Permette una connessione a linee telefoniche (non a Internet) tradizionali. Ha una velocità massima di trasmissione di 256.000 baud. È utilizzato per determinare quando un dispositivo esterno collegato su una porta seriale desidera colloquiare con la nostra applicazione, inviandoci dei dati.

MaskEdBox (Microsoft MaskEdit Control 6.0)
È una TextBox che obbliga l'utente ad inserire particolari dati, evitando al programmatore l'utilizzo di lunghe procedure per il controllo e la validazione dell'input.
La proprietà *Mask* imposta la maschera per l'inserimento dei dati, i valori ammessi sono simili a quelli della funzione Format.
MaskEdBox1.Mask = "##-???-####" '01-apr-2002
MaskEdBox1.Mask = "Nick: >aaaaaaa" 'scrive tutto in maiuscolo
MaskEdBox1.Mask = "\Telefono ####-######"

MMControl (Microsoft Multimedia Control 6.0)
Permette di controllare i dispositivi audio o video collegati al PC. Sono usati i comandi MCI (Media Control Interface)
Private Sub Form_Load()
MMControl1.DeviceType = "WaveAudio"

```
    MMControl1.FileName = "C:\windows\media\tada.wav"
    MMControl1.Command = "open"
End Sub
```

Individuare il nome del file eseguibile

```
Dim percorso As String
Percorso = App.EXEName
```

Acquisire tutte le impostazioni dell'ambiente (environment)

La configurazione del sistema operativo fa uso di variabili d'ambiente.

```
Private Sub Command1_Click()
        Dim x As Integer
        x = 1
        While Environ (x) <> ""
                List1.AddItem Environ (x)
                X = x + 1
        Wend
End Sub
```

I vettori di controlli

Si definisce un vettore i cui elementi sono rappresentati da controlli. Rappresentano un'eccezione rispetto al modo usuale di nominare gli oggetti, perchè i controlli che fanno parte dello stesso vettore sono tutti identificati con lo stesso nome e tra loro sono distinti tramite un indice numerico (Index). In fase di progettazione, il programmatore può definirli in due modi.

1. Duplicando un controllo già esistenti (Copia e Incolla).
2. Definendo due o più controlli dello stesso tipo con la stessa proprietà Name.

La scrittura del codice relativo ai singoli oggetti cambia perchè la routine sarà una sola, associata all'unico oggetto che è il vettore di controlli, ma le istruzioni relative ad ogni singolo oggetto possono essere distinte tra loro mediante la struttura Select Case o la struttura If.

Tastiera

Per agevolare la lettura del codice è associato ad ogni tasto sulla tastiera una costante definita nella classe *KeyCodeConstants*.

Impostare la proprietà *Form1.KeyPreview = True* e digitare il codice.

```
Private Sub Form_KeyPress(KeyAscii As Integer)
    Form1.Print "Codice ASCII del tasto premuto: ", KeyAscii
End Sub
```

GESTIONE DI FILE DI RISORSE

Le risorse possono essere:

1. predefinite dell'IDE quali controlli grafici, menu, icone, bitmap;
2. create appositamente dal programmatore e memorizzate in un file di risorse.

Nell'IDE, per creare un file di risorse (.RES) si usa lo strumento Editor di risorse.
Questi deve essere caricato nell'IDE con *Aggiunte/Gestione aggiunte/Editor risorse VB 6*, terminata questa operazione l'editor è disponibile nel menu *Strumenti/Editor risorse*.
L'editor mette a disposizione una serie di risorse di tipo diverso.

Tipo di risorsa	Utilizzata per	Richiamo
Tabelle stringhe	Applicazioni internazionali	LoadResString (ID)
Cursore, Icona, Bitmap	Gestione immagini	LoadResPicture (ID,tipo)
Creata dall'utente	…	LoadResData (ID)

Un'applicazione richiama una risorsa mediante un **identificatore di risorse** (Resource IDentifier) univoco definito durante l'inserimento della risorsa nel file. Ciascun ID ha una propria riga e ciascuna riga può avere più colonne.
Dopo aver aggiunto tutte le risorse e averle salvate in un file di risorse, il Resource Editor compila automaticamente le risorse e non è più necessario utilizzare il Resource Compiler.
L'uso del file di risorse (al massimo 64 KB) permette di evitare la modifica del codice sorgente dell'applicazione, se rimangono inalterati i riferimenti agli identificatori delle risorse, ogniqualvolta si modificano le risorse stesse.
Progettare un'applicazione bilingue italiano-inglese (Bilingue.vbp)
Editor risorse: inserire una nuova colonna di nome Inglese (Stati Uniti), inserire le righe di nome 102 fino a 104 e salvare il file di risorse con il nome di bilingue.res.
Editor di menu: progettare il menu

SENZA.VBP

Costruzione di un'applicazione priva d'interfaccia grafica: per svolgere operazioni non visibili all'utente. **File/Rimuovi File** e **Inserisci/Modulo**, inserire il codice in Oggetto (generale) e Routine Main.

```
Public Sub Main()
        Beep
        MsgBox ("Main")
End Sub
```

La routine Main () è eseguita senza che sia necessaria la presenza di un form. Utile per applicazioni che comunicano con altre applicazioni, inoltre con *Load* e S*how* si possono visualizzare finestre in applicazioni che in origine non ne posseggono.

CALC.VBP

Inserire nel form tre TextBox: primo addendo, secondo addendo e somma.

Oggetto	Proprietà	Impostazione
Form1	Caption	Calcolatrice
Label1	Caption	+
	BorderStyle	1
Command1	Caption	=

Doppio clic su CommandButton, inserire il codice

```
Private Sub Command1_Click ()
    Dim op1 As Single
    Dim op2 As Single
    op1 = Val(Text1.Text)
    op2 = Val(Text2.Text)
    Text3.Text = Format$(op1 + op2, "###,###,###")
End Sub
```

L'utente può passare da una casella di testo all'altra con TAB o con il mouse.
Quando un controllo contiene il punto di inserimento si dice che possiede il **focus**.
La casella di somma non deve ricevere il focus per questo T*abStop* = False (rimozione dell'ordine di tabulazione), si usa la proprietà *TabIndex* per l'impostazione dell'ordine di tabulazione.
L'evento *GotFocus* si verifica quando l'evidenziazione è spostata su un oggetto.
L'evento *LostFocus* si verifica quando l'evidenziazione è rimossa dall'oggetto.
L'utente non usa <CR>, allora il pulsante "uguale" diventa quello di default ed è indicato con bordo nero spesso, per attivarlo D*efault* = True.
Formattazione dell'output

```
Format$ (1234.56,"###.#")                 '= 1234.5
Format$ (1234.56,"00000.000")             '= 01234.560
Format$ (1234.56,"###,###.0")             '= 1,234.56
Format$ (1234.56,"$#,000.00")             '= $1,234.56
Format$ (Now,"hh:mm:ss)                   '= "20.18.23"
Format$ (Now,"dd/mm/yy")                  '= "27/01/94"
Format$ (Now,"dddd, dd - mmmm - yyyy")    '= "mercoledì, 27 - gennaio - 1994"
```

EDITOR.VBP

Utilizziamo delle nuove proprietà della TextBox, il testo può essere lungo al massimo 64K con più righe ed a capo automatico.

Oggetto	Proprietà	Impostazione
Form1	Caption	Editor
Text1	Name	PadText
	Multiline	True
	ScrollBars	2 - Vertical

Label1	Caption	+
	BorderStyle	1
CommandButton	Name	ClearButton
	Caption	&Cancella Tutto (la & sta per ALT+C)
CommandButton	Name	CutButton
	Caption	&Taglia (la & sta per ALT+T)
CommandButton	Name	PasteButton
	Caption	&Incolla (la & sta per ALT+I)

Quando si cancella la finestra di editor il focus è mantenuto sul pulsante, per ritornare all'editor clic o TAB: è poco professionale, Il codice deve essere:

```
Private Sub ClearButton_Click ()
    PadText.Text = ""
    PadText.SetFocus                    'ritorna il focus all'editor
End Sub
```

Prima di tagliare, il testo deve essere salvato negli appunti, il codice deve essere:

```
Private Sub CutButton_Click()
    Clipboard.Clear
    Clipboard.SetText PadText.SelText   'stringa con i caratteri tagliati
    PadText.SelText = ""
    PasteButton.Enabled = True
    PadText.SetFocus
End Sub
Private Sub PasteButton_Click()
    PadText.SelText = Clipboard.GetText
    PadText.SetFocus
End Sub
```

Il pulsante Incolla quando, si entra in editor, deve essere disabilitato (grigio) per le modalità Windows, allora bisogna settare la proprietà *Enable* = False e nel codice di Taglia l'istruzione: *PasteButton.Enabled = True.* Vale, all'inizio la stessa regola, per Taglia e Cancella Tutto, per questo in PadText si può sfruttare l'evento *Change* che è attivato quando si modifica la casella di testo. *Enabled* deve essere impostato per tutti e tre i pulsanti. Il codice nella casella di testo deve essere:

```
Private Sub PadText_Change ()
    CutButton.Enabled = True
    ClearButton.Enabled = True
End Sub
```

CLIPBOARD (APPUNTI)

Strumento che consente di trasferire testo o disegni tra le varie applicazioni, grazie alle operazioni di Taglia, Copia e Incolla. L'oggetto è *Clipboard* e i metodi sono:

Clear	Cancella gli Appunti
GetText	Preleva il testo negli Appunti
GetData	Preleva l'immagine negli Appunti
SetData	Incolla l'immagine negli Appunti
GetFormat	Preleva il formato (testo o immagine) negli Appunti
SetText	Incolla il testo negli Appunti

```
'Form con un controllo Picture e un CommadButton
Private Sub Command1_Click()
  If Clipboard.GetFormat(1) Then
    MsgBox "Testo *.txt"
  ElseIf Clipboard.GetFormat(2) Then
    MsgBox "Bitmap *.bmp"
    Picture1.Picture = Clipboard.GetData(2)
```

```
   ElseIf Clipboard.GetFormat(3) Then
      MsgBox "Metafile *.vmf"
      Picture1.Picture = Clipboard.GetData(3)
   ElseIf Clipboard.GetFormat(8) Then
      MsgBox "Dib *.dib"
   End If
End Sub
```

EDITOR1.VBP

Progettare i **menu a tendina** per i form nell'Editor di menu.
È possibile visualizzare questa finestra scegliendo **Strumenti/Editor di menu...CTRL+E** oppure facendo clic sul pulsante Editor di menu sulla barra degli strumenti.
I menu sono costituiti da un titolo del menu, voci di menu e barre di separazione.
Il menu è un controllo, come una casella di testo o un pulsante di comando.
In modo analogo agli altri controlli, il menu dispone di un insieme predefinito di proprietà ed eventi.
Per definire le modalità di risposta di ciascun comando di menu ad un evento Click, scrivere una routine di eventi per ciascun comando.
Per esempio, su File/Esci il codice deve essere:

```
Private Sub ExitItem_Click()
        End
Private Sub End
```

Per la creazione di menu, attenersi alle seguenti indicazioni standard.

- ✓ Raggruppare i comandi correlati in un menu in modo logico per gli utenti dell'applicazione. Gli utenti che hanno familiarità con Windows, ad esempio, si aspettano che i comandi Nuovo, Apri e Chiudi siano contenuti nel menu File. Per un esempio, fare riferimento alle applicazioni per Windows disponibili.
- ✓ Nei menu contenenti molti comandi, separare i gruppi di comandi correlati con una barra di separazione. Le barre di separazione sono create utilizzando un singolo trattino (-) nella casella "Caption" dell'Editor di menu.

Caption	Name	Shortcut
&File	FileMenu	
&Apri	LoadItem	
Sa&lva con nome	SaveItem	
-	Separator	
&Esci	ExitItem	
&Modifica	ModificaMenu	
Tag&lia	CutItem	CTRL+X
I&ncolla	PasteItem	CTRL+V
&Cancella tutto	ClearItem	CTRL+C

Adesso bisogna trasferire il codice dai pulsanti ai menu, doppio clic sul pulsante Taglia, sostituire *CutButton_Click()* con *CutItem_Click()* e selezionare nella finestra di codice l'oggetto *CutItem*; stesso procedimento per gli altri due.
I menu non essendo oggetti come i pulsanti, il focus appartiene sempre alla Textbox, quindi SetFocus si può cancellare in tutti e tre i casi. I tre pulsanti sono inattivi all'avvio dell'editor, quindi pure le voci del menu Modifica, si usa la proprietà *Enabled*, basta modificare in *PadText_Change():*

CutButton.Enabled con *CutItem.Enabled*
ClearButton.Enabled con *ClearItem.Enabled* ed in *CutItem_Click()*
PasteButton.Enabled con *PasteItem.Enabled*.

Adesso bisogna rimuovere i pulsanti e rendere inattive (grigie) le voci del menu Modifica. Selezionare **Strumenti/Editor di menu...CTRL+E** togliere il segno di spunta di *Enabled* in tutti e tre i casi.

Selezione dei font da menu, in una Textbox si può usare solo un tipo di font per volta; aggiungere il menu:

Caption	Name	Index
F&ont	FontMenu	
Courier	FFF	0
Termianl	FFF	7

per i font non conviene avere un nome diverso, ma una **Matrice di controlli** (è costituita da un gruppo di controlli dello stesso tipo, con lo stesso nome e le stesse routine di eventi, al massimo 32767; vantaggi: minore numero di risorse e stesso codice per più controlli); a *Caption* diversi (Helv, Courier, Symbol) corrisponde lo stesso *Name* = FFF. Visual BASIC non assegna, come nei pulsanti, Index in modo automatico ma bisogna usare la casella *Index* mettendo i valori da zero a sette. Clic su una voce qualsiasi per il codice:

```
Private Sub FFF_Click(Index As Integer)
    Dim i As Integer
    Select Case Index
        Case 0
            PadText.FontName = "Courier"
        Case 1
            PadText.FontName = "Helv"
        Case 2
            PadText.FontName = "Roman"
        Case 3
            PadText.FontName = "Modern"
        Case 4
            PadText.FontName = "Script"
        Case 5
            PadText.FontName = "Symbol"
        Case 6
            PadText.FontName = "System"
        Case 7
            PadText.FontName = "Terminal"
    End Select
    For i = 0 To 7
        FFF(i).Checked = False
    Next i
    FFF(Index).Checked = True
End Sub
```

Per sapere quale font è attivo nel menu occorre il **segno di spunta**: per le caselle di testo il font attivo è Helv, per spuntarlo selezionare *Checked*.

Per aggiungere al menu i **tasti di scelta rapida** basta anteporre la & alla lettera desiderata.

Ogni tasto deve essere esclusivo all'interno del suo livello, in pratica due nomi di menu della stessa barra non possono usare lo stesso tasto e due voci del menu non devono avere la stessa lettera.

Per aggiungere al menu i **tasti scorciatoia** basta evidenziare un menu e selezionare *Shortcut* e scegliere per esempio CTRL A.

La proprietà Visible delle voci di menu permette di nascondere le voci stesse.

Controlli per lo stato dei tasti

Inserire nel progetto **Strumenti/Controlli aggiuntivi.../Microhelp key State Control** (sono indicatori che mostrano lo stato dei tasti) e per ogni controllo impostare la proprietà *Style* in questo modo:

0 Caps Lock (Bloc Maiusc);

1 Num Lock (Bloc Num);

2 Insert (Ins);
3 Scroll Lock (Bloc Scorr).

AGENDA.VBP

È possibile inserire voci di menu con *Load* ed eliminarle con *Unload*, purché siano parte di una matrice di controlli, per questo si deve avere sempre almeno una voce, affinché le altre siano aggiunte a questa. Si disegna il menu.

Caption	Name	Index
&File	FileMenu	
Aggiungi nome corrente	AddNameItem	
-	SeparatorItem	
(blank)	NNN	0
&Esci	ExitItem	

Clic su Aggiungi nome corrente, in Oggetto (generale) e Routine (dichiarazioni) e digitare:

```
Private Names(1 To 10)  As String
Private Numbers(1 To 10) As String
```

Clic su AddNameItem, inserire

```
Private Sub AddNameItem_Click ()
    Static n As Byte                ‘numero di voci del menu
    n = n + 1                       ‘aggiunge una voce
    Load NNN(n)
    NNN(0).Visible = False
    NNN(n).Caption = Text1.Text
    NNN(n).Visible = True
    Names(n) = Text1.Text
    Numbers(n) = Text2.Text
End Sub
```

Dopo aver aggiunto i nomi al menu, è possibile con un clic visualizzare Nome e Numero, selezionare *NNN_Click* ed inserire il codice:

```
Private Sub NNN_Click (Index As Integer)
    Text1.Text = Names(Index)
    Text2.Text = Numbers(Index)
End Sub
```

POPUP.VBP

I menu **Pop-Up** (**menu di scelta rapida**, **menu contestuali**) sono menu visualizzati nel form quando si fa clic con il pulsante destro del mouse e sono indipendenti dai menu a tendina e dalla barra degli strumenti.

È possibile visualizzare un solo menu pop-up alla volta.

Per creare un menu che non è visualizzato nei menu a tendina, bisogna rendere invisibile il nome del menu in fase di progettazione (casella “Visible”). Visual BASIC mette a disposizione automaticamente un menu pop-up per il controllo TextBox.

Se si crea un menu pop-up personalizzato, è visualizzato per primo il menu di sistema e dopo il menu personalizzato.

La **Barra degli Strumenti**, integra i menu a tendina, e fornisce una rappresentazione grafica dei comandi dell’applicazione.

È inoltre possibile utilizzare le descrizioni dei comandi, che consentono di visualizzare il nome di ciascun pulsante al semplice passaggio del mouse sul pulsante stesso.

L’evento click su questi pulsanti deve essere uguale all’evento click sul corrispondente elemento del menu a tendina.

Caption	Name	Index	Visible
Hidden	mnu200		non spuntato
Ripristina formato	mnu201	0	

```
Minimize        mnu202    1
Maximize        mnu203    2
Private ix, iy As Single                    'coordinate dell'info_icona
Private nome_info As String                 'testo dell'info
Private MsgLin(10) As String
Private MsgLine(10) As String
Private Sub sposta_info()
    Label1.Top = iy
    Label1.Left = ix
    Label1.Visible = True
    Label1.Caption = nome_info
End Sub
Private Sub cmdToolBar_Click(Index As Integer)
    mnu101_Click (Index)
End Sub
Private Sub cmdToolBar_MouseMove(Index As Integer, Button As Integer, Shift As
Integer, X As Single, Y As Single)
    panMsgLine.Caption = MsgLine(Index)
    ix = X + cmdToolBar(Index).Left
    iy = Y + 490
    nome_info = MsgLin(Index)
    sposta_info
End Sub
Private Sub Form_Load()
    Dim i As Integer
    Move (Screen.Width - Width) \ 2, (Screen.Height - Height) \ 2
    mnu300.Caption = Chr(8) & mnu300.Caption
    MsgLine(0) = "Gestione anagrafiche Clienti e Fornitori"
    MsgLine(1) = "Schede carico e scarico di Magazzino"
    MsgLine(2) = "Bolle, Fatture e Note di credito"
    MsgLine(3) = "Chiusure: giornaliera - mensile - annuale"
    MsgLine(4) = "Elaborazioni Statistiche"
    MsgLine(5) = "Procedura di Back-Up e Restore"
    MsgLine(7) = "Uscita dal programma"
    MsgLin(0) = "Gestione"
    MsgLin(1) = "Magazzino"
    MsgLin(2) = "Fatture"
    MsgLin(3) = "Chiusure"
    MsgLin(4) = "Elaborazioni"
    MsgLin(5) = "Back-Up"
    MsgLin(7) = "Uscita"
End Sub
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y
As Single)
    Label1.Visible = False
End Sub
Private Sub mnu101_Click(Index As Integer)
    If Index = 7 Then End
    MsgBox MsgLine(Index), 64, "Procedura da attivare"
End Sub
Private Sub mnu201_Click(Index As Integer)
    frmMain.WindowState = Index
End Sub
Private Sub panToolBar_MouseMove(Button As Integer, Shift As Integer, X As
```

```
Single, Y As Single)
        panMsgLine.Caption = "Clickare il tasto destro sopra la figura dell'omino..."
End Sub
Private Sub picMain_MouseDown(Button As Integer, Shift As Integer, X As Single,
Y As Single)
If Button = 2 Then PopupMenu mnu200, 2
End Sub
```

Evento MouseDown

È generato quando l'utente preme semplicemente un pulsante del mouse in un determinato punto del form, della PictureBox e della Label; per l'evento *Click*, invece, si deve premere e rilasciare il pulsante.

La routine riceve i seguenti argomenti: x, y posizione del puntatore espressa in twip; Button quale pulsante è stato premuto codificandolo in questo modo.

VALORE	BINARIO	SIGNIFICATO
1	0000000000000001	è stato premuto il pulsante sinistro
2	0000000000000010	è stato premuto il pulsante destro
4	0000000000000100	è stato premuto il pulsante centrale

Shift indica se, mentre era premuto il pulsante del mouse, era premuto anche il tasto CTRL, il tasto MAIUSC o il tasto ALT.

VALORE	BINARIO	SIGNIFICATO
0	0000000000000000	non è premuto né MAIUC né CTRL
1	0000000000000001	è premuto il tasto MAIUSC
2	0000000000000010	è premuto il tasto CTRL
4	0000000000000100	è premuto il tasto ALT

```
Private Sub Form_MouseDown (Button As Integer, Shift As Integer, x As Single, y As
Single)
    Text1.Text = Str$(x)
    Text2.Text = Str$(y)
    Select Case Button
        Case 1
            Text3.Text = "Pulsante sinistro"
        Case 2
            Text3.Text = "Pulsante destro"
    End Select
End Sub
```

Evento MouseUp

È generato quando un pulsante del mouse è rilasciato.

Evento MouseMove

È generato quando si preme un pulsante e si muove il mouse: valori restituiti per Button.

VALORE	BINARIO	SIGNIFICATO
0	0000000000000000	non è premuto nessun pulsante
1	0000000000000001	è premuto solo il pulsante sinistro
2	0000000000000010	è premuto solo il pulsante destro
3	0000000000000011	sono premuti i pulsanti destro e sinistro
4	0000000000000100	è premuto solo il pulsante centrale
5	0000000000000101	sono premuti i pulsanti centrale e sinistro
6	0000000000000110	sono premuti i pulsanti centrale e destro
7	0000000000000111	sono premuti tutti e tre i pulsanti

Stili del puntatore del mouse, è la proprietà *MousePointer*. Per visualizzare un puntatore personalizzato:

```
Screen.MousePointer = 99
```

Screen.MouseIcon = LoadPicture (“c:\windows\pointer\pippo.ico”)
Sub Form_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
Form1.Print x, y
End Sub

CLIPPER.VBP

L’esempio illustra la **determinazione del controlo attivo**; la proprietà *ActiveControl* consente di determinare quale sia il controllo attivo e di fare riferimento ad esso nel modo consueto.
L’applicazione possiede due caselle di testo e due menu Copia ed Incolla, quando l’utente seleziona del testo è necessario determinare a quale casella l’utente si sta riferendo.
A questo scopo, va verificato qual è il controllo attivo, in pratica a quale controllo è assegnato il focus.

```
Private Sub CopyItem_Click ()
    Clipboard.SetText Screen.ActiveControl.SelText
End Sub
Private Sub PasteItem_Click ()
    Screen.ActiveControl.SelText = Clipboard.GetText()
End Sub
```

È disponibile anche *ActiveForm* che consente di determinare quale form è attivo, contiene il valore *Me* per il form corrente.
Inoltre, si può verificare di quanti form è dotata l’applicazione con *Forms.Count*.

CASELLA.VBP

Oggetto	Proprietà	Impostazione
Form1	Caption	Windows Shell
Label1	Caption	About

Inserire il codice.

```
Private Sub Label1_Click ()
    Dim RetVal As Byte
    RetVal=MsgBox("Questo pulsante visualizza informazioni di about...", 65, "About")
End Sub
```

Si usa la funzione:

```
RetVal = Shell (comando$ [,tipofinestra$])
```

dove gli argomenti hanno il seguente significato:

comando$	nome dell’applicazione EXE, BAT, COM, PIF		
tipofinestra$	opzioni di avvio della finestra	1	finestra normale, con focus
		2	ridotta, con focus
		3	ingrandita, con focus
		4	normale, senza focus
		7	ridotta, senza focus
RetVal	ritorna l’ID dell’applicazione.		

Disegnare il menu.

Caption	Name
&File	FileMenu
Ese&gui...	RunItem
-	Separator
&Esci	ExitItem

Siccome ci sono i tre punti l’utente si aspetta una finestra di dialogo dove digitare il nome dell’applicazione da eseguire. Clic su *Esegui...*, inserire il codice:

```
Private Sub RunItem_Click ()
    Dim RetVal As String
     Dim pippo As Long
```

```
                    'se si sceglie annulla dalla InputBox$ ritorna una stringa nulla
        RetVal = InputBox$("Applicazione da eseguire:", "Esegui...", "Notepad")
        If RetVal <> "" Then
            pippo = Shell(RetVal, 1)
        Else
            Exit Sub
        End If
    End Sub
```

Un'applicazione diventa attiva non appena si vede il titolo della finestra annerito.
Da questo momento in poi qualsiasi evento sia generato, l'applicazione è in grado d'intercettarlo e rispondere di conseguenza.
Per attivare un'applicazione che è stata lanciata tramite Visual BASIC si usa l'istruzione: *AppActivate variabile*, dove variabile è l'ID restituito dalla Shell.
È ovvio che questa variabile non deve perdere il suo valore tra una chiamata e l'altra, per evitare questo è possibile agire in due modi.

1. Dichiarare la variabile Static.
2. Dichiarare le variabile globale.

Dopo aver reso l'applicazione esterna attiva si può pensare d'inviarle qualsiasi sequenza di tasti, come se si fosse un utente davanti alla tastiera che preme una sequenza di tasti, mediante l'istruzione *SendKeys stringa*, [wait].
Dove *stringa* rappresenta la sequenza di caratteri da inviare all'applicazione e *wait* può assumere i valori True o False.
Nel primo caso il controllo non è restituito all'applicazione fino a quando i tasti inviati non sono stati elaborati dall'applicazione esterna, nel secondo caso il controllo è restituito subito all'applicazione chiamante.
Per simulare l'invio di un tasto s'indica nella stringa il carattere stesso, i tasti +, ^, %, ~, () devono essere racchiusi tra parentesi graffe {%}.
I tasti che possono essere premuti insieme ad altri sono SHIFT (+), ALT (%) e CTRL (^). Per esempio, premere SHIFT+E+C si scrive "+(EC)".
Progettare un'applicazione (Shell.vbp) per lanciare il Blocco note e la Calcolatrice.
In Blocco note deve comparire la stringa "Sto per eseguire il seguente conto: ", attenda tre secondi e scriva "3 + 5".
Dopo due secondi si deve vedere l'operazione sulla Calcolatrice e nel momento in cui compare il risultato, nel Blocco note deve comparire la scritta "Eseguito!!!"

```
Private Sub Command1_Click()
    Static abilitatocalc As Boolean, abilitatonote As Boolean
    Dim idnote As Long, idcalc As Long, tempo As Long
    tempo = Timer()
    idnote = Shell("c:\windows\notepad.exe", 1)
    Do While Timer() < tempo + 3
    Loop
    SendKeys "Ben arrivati in Blocco note {ENTER}", True
    SendKeys "Sto per eseguire il seguente conto: ", True
    Do While Timer() < tempo + 5
    Loop
    AppActivate idnote
    SendKeys "{ENTER}3 {+} 5 =  ", True
    Do While Timer() < tempo + 8
    Loop
    idcalc = Shell("c:\windows\system32\calc.exe", 1)
    AppActivate idcalc
    Do While Timer() < tempo + 10
    Loop
    SendKeys "3", True
```

```
    Do While Timer() < tempo + 11
    Loop
    SendKeys "{+}", True
    Do While Timer() < tempo + 12
    Loop
    SendKeys "5", True
    Do While Timer() < tempo + 13
    Loop
    SendKeys "{ENTER}", True
    Do While Timer() < tempo + 15
    Loop
    AppActivate idnote
    Do While Timer() < tempo + 16
    Loop
    SendKeys "Eseguito!!!", True
End Sub
```

CASELLA1.VBP

Utilizza le **finestre multiple**, a partire da *casella.mak* si inserisce una nuova finestra, si crea un nuovo form con **Inserisci/Form**.

Oggetto	Proprietà	Impostazione
Form1	Caption	Esegui...
	Name	RunDialog
	ControlBox	False
	BorderStyle	3
TextBox		
CommandButton	Name	OKButton
	Caption	OK
	Default	True
	TabStop	True
CommandButton	Name	CancelButton
	Caption	Annulla
	Cancel	True

Siccome è una finestra di dialogo di tipo **modal**, l'utente non potrà effettuare nessuna operazione prima di aver soddisfatto la richiesta di questa finestra.
L'evento che provoca la visualizzazione della finestra di dialogo è *RunItem_Click()* della voce File/Esegui... doppio clic su questa opzione compare il codice dell'istruzione *shell* inserito in precedenza, adesso invece si deve visualizzare la finestra di dialogo:

```
Private Sub RunItem_Click ()
    RunDialog.Show 1
End Sub
```

il form RunDialog è visualizzato anche se non è in memoria perché il metodo *Show* lo carica automaticamente. L'istruzione per visualizzare e caricare un nuovo form è:

```
Form1.Show [modal%]
```

- *modal* = 0, l'utente può usare altri Form, mentre Form1 è visualizzato;
- *modal* = 1, tutte gli altri form dell'applicazione sono inattivi: è di tipo modal.

Doppio clic su OK ed inserire il codice:

```
Sub OKButton_Click ()
    Dim RetVal As Intger
    RetVal = Shell(Text1.Text, 1)
    Unload RunDialog
End Sub
```

Dopo aver eseguito la Shell, si deve eliminare la visualizzazione della finestra di dialogo;

in questo modo, quando l'applicazione Windows si conclude, si ritorna a Windows Shell e non alla finestra di dialogo. Doppio clic su Annulla, inserire il codice:

```
Sub CancelButton_Click ()
    'RunDialog.Hide          elimina la visualizzazione del form
    Unload RunDialog         'scarica il form dalla memoria
End Sub
```

PANNELLO.VBP

Serve a personalizzare gli aspetti di un'applicazione. Disegnare il menu.

Caption	Name	Shortcut
&File	FileMenu	
&Pannello di controllo...	ControlPanelItem	CTRL+P
-	Separator	
&Esci	ExitItem	

Inserisci/Form con i seguenti oggetti.

Oggetto	Proprietà	Impostazione
Form1	Caption	Pannello di controllo
	Name	ControlPanel
TextBox1	Name	NewCaption
	Caption	(si modifica il titolo in Form1)
Label1	Caption	Titolo dell'applicazione
CommandButton1	Caption	OK
	Name	OKButton
CommandButton	Caption	Annulla
	Name	CancelButton

Doppio clic su Annulla ed inserire il codice:

```
Private Sub CancelButton_Click ()
    Unload ControlPanel
End Sub
```

Doppio clic su OK ed inserire il codice:

```
Private Sub OKButton_Click ()
    Form1.Caption = NewCaption.Text
    Form1.Height = NewHeight.Value
    Form1.Width = NewWidth.Value
    'si deve riportare il colore impostato nello sfondo della finestra principale.
    Form1.BackColor = RGB(NewRed.Value, NewGreen.Value, NewBlue.Value)
    Unload ControlPanel
End Sub
```

L'unità di misura di default in Visual BASIC è il **twip**, che equivale ad 1/1440 di pollice.

Per modificare l'altezza e la larghezza della finestra si usano le proprietà *Form1.Height* e *Form1.Width* leggendo i valori attuali tramite *Val()*.

Tuttavia, non è semplice esprimere la misura in twip, si possono usare allora le **barre di scorrimento** con le loro proprietà.

1. *Min*, è il valore numerico che si assegna all'estremità superiore sinistra, tutte le proprietà possono assumere valori compresi tra 0 e 32767.
2. *Max*, è il valore che si assegna all'estremità inferiore destra.
3. *Value*, è il valore corrente che corrisponde alla casella che si muove all'interno della barra, Min <= Value <= Max.
4. *LargeChange*, indica l'entità della modifica subita da *Value* ogni volta che l'utente fa clic nella barra.
5. *SmallChange*, indica l'entità della modifica subita da *Value* ogni volta che l'utente fa clic sulle frecce.

L'evento principale correlato alle barre di scorrimento è l'evento *Change*, che si verifica

ogni volta che la casella è spostata; tuttavia, in questo caso si vuole che le modifiche abbiano effetto solo quando l'utente fa clic si OK.
Posizionare nel form i seguenti oggetti.

Oggetto	Proprietà	Impostazione
Label	Name	Label2
	Caption	Nuova altezza
VScrollBar	Name	NewHeight
	Min	1440
	Max	7200
	LargeChange	1000
	SmallChange	500
Label	Name	Label3
	Caption	Nuova larghezza
VScrollBar	Name	NewWidth
	Min	2880
	Max	10080

Le dimensioni della finestra sono:

	Pollici	Twip
Minima	1" * 2" (altezza * larghezza)	1440 * 2880
Massima	5" * 7"	7200 * 10080

Doppio clic su File/Pannello di controllo... ed inserire il codice:

```
Private Sub ControlPanelItem_Click ()
    ControlPanel.NewCaption.Text = Form1.Caption
    ControlPanel.NewHeight.Value = Form1.Height
    ControlPanel.NewWidth.Value = Form1.Width
''preleva il colore di sfondo della finestra principale e lo carica nel pannello di controllo
    ControlPanel.NewColor.BackColor = Form1.BackColor
    ControlPanel.Show 1
End Sub
```

Quando l'utente seleziona questa voce, si devono caricare i controlli del pannello di controllo con le impostazioni correnti della finestra e visualizzare quindi il pannello sullo schermo.
Se si usano le barre di scorrimento si possono reimpostare le dimensioni della finestra; ciò non costituisce un vantaggio significativo, dal momento che si possono trascinare i bordi, ma ha un valore esemplificativo.
I **colori** sono determinati da tre fattori cromatici indipendenti: il rosso, il verde ed il blu. Ogni valore di colore varia da 0 a 255, tutti e tre sono poi sommati in un Long e la somma è inserita nelle proprietà *Backcolor*, *Forecolor*.
Visual BASIC mette a disposizione i seguenti metodi:

1. la funzione *RGB()*, riceve tre parametri interi compresi tra 0 e 255 che indicano la quantità rispettivamente di rosso verde e blu (in ordine contrario alla codifica esadecimale); RGB (255,0,0) rosso, RGB (0,255,0) verde, RGB (0,0,0255) blu.
2. la funzione *QBColor()*, riceve un parametro compreso tra 0 e 15, rappresentante un colore (0 nero, 1 blu, 2 verde, 4 rosso, 7 bianco);
3. l'impostazione diretta in un Long: da &H000000& a &HFFFFFF& le ultime due cifre rappresentano la quantità di rosso, le due centrali la quantità di verde, le prime due la quantità di blu (&H000000& assenza di colori è nero, &HFFFFFF& tutti i colori presenti è bianco, &HFF0000& blu intenso, &H00FF00& verde intenso)
4. usare le costanti Visual BASIC (*vbBlack* nero, *vbBlue* blu, *vbGreen* verde, *vbRed* rosso)
5. le proprietà *Forecolor* e *Backcolor.*

Il metodo: *[oggetto.]Point (x,y)* ritorna il valore in un Long del colore.
Posizionare nel form i seguenti oggetti.

Oggetto	Proprietà	Impostazione

VScrollBar	Name	NewRed, NewGreen, NewBlue
	Min	0
	Max	255
	LargeChange	20
	SmallChange	10
Label	Name	NewColor

La label NewColor offre all'utente un riscontro visivo immediato del colore.
Doppio clic su Rosso (Verde e Blu) ed inserire il codice:

```
Sub NewRed_Change ()
    NewColor.BackColor = RGB(NewRed.Value, NewGreen.Value, NewBlue.Value)
End Sub
```

Questa routine è richiamata ogni volta che *NewRed.Value* subisce una modifica.
Sebbene si possa leggere questo valore quando è premuto OK e modificare *Form1.BackColor*, si può usare questo valore anche per mostrarlo in NewColor.

MDI.VBP

L'interfaccia **MDI** (*Multiple Document Interface*) è stata progettata appositamente per applicazioni orientate a documenti.
L'altro tipo d'interfaccia è la **SDI** (*Single Document Interface*).
L'icona di un form secondario ridotto a icona non è visualizzata nel desktop, ma nel form MDI.
Quando un form secondario è ingrandito, la relativa didascalia è unita alla didascalia del form MDI e visualizzata nella barra del titolo del form MDI.
Sono varie finestre presenti contemporaneamente sullo schermo, è possibile spostare la finestra nella finestra principale ma non al di fuori di essa, è consentito l'uso di un solo form MDI per applicazione.

Inserisci/Form MDI

Oggetto	Proprietà	Impostazione
Form1	MDIChild	True
Form2	MDIChild	True

Quando si esegue l'applicazione, inizialmente compare solo Form1; per visualizzare Form2 si aggiunga il seguente codice:

```
    Sub Form_Click ()
        Form2.Show
    End Sub
```

Ora quando si fa clic su Form1 compare Form2.

SOVRAP.VBP

L'esempio illustra il **passaggio di form alle routine**, si tratta di un'opzione molto utile se si hanno numerosi form ai quali si vogliono applicare le stesse istruzioni o che si vogliono coordinare, in pratica quante più finestre si devono gestire nello stesso modo tanto più utile è il raggruppamento del codice di gestione in una sola routine.
L'applicazione deve gestire quattro form, facendo clic l'utente deve poter sovrapporre le finestre ed assegnare a tutte le stesse dimensioni. Si può cominciare verificando che tutte le finestre siano visualizzate sullo schermo. Form1 deve essere il form iniziale.

```
    Private Sub Form_Load ()
        Form2.Show
        Form3.Show
        Form4.Show
    End Sub
```

A questo punto si può aggiungere il CommandButton Sovrapposte.

```
    Sub Command1_Click ()
        Cascade Form1
```

```
        Cascade Form2
        Cascade Form3
        Cascade Form4
    End Sub
```

La routine *Cascade()* dispone le finestre, per passare i form si usa la dichiarazione *As Form*; dato che i form devono essere sovrapposti, bisognerà mantenere la posizione dell'angolo superiore sinistro di ciascuno di essi fra una chiamata e l'altra ecco perché le coordinate devono essere *Static*.

```
    Private Sub Cascade (TheForm As Form)
        Static TopX, TopY As Integer
        TheForm.Width = Screen.Width / 4
        TheForm.Height = Screen.Height / 4
        TheForm.Move TopX, TopY
        TopX = TopX + Screen.Width / 10
        TopY = TopY + Screen.Height / 10
    End Sub
```

Per modificare il titolo del form si usa il seguente codice.

```
    Private Sub CambiaNome (TheForm As Form)
        TheForm.Caption = "Hello, World"
    End Sub
```

A questo punto, è necessario passare il Form1 alla routine, con questo codice.

```
    Sub Form_Click ()
        CambiaNome Me
    End Sub
```

Non è possibile utilizzare *CambiaNome()* Form1 perché potrebbero essere in esecuzione più copie di Form1, si usa la parola riservata *Me* che si riferisce al form corrente.

DESKTOP.VBP

Lo scopo è quello di consentire all'utente di **fare clic su un controllo e di trascinarlo**.
Si utilizza la proprietà *DragMode* e l'evento *DragDrop* del form.
I nomi delle applicazioni sono inseriti nelle Label invece che in TextBox (ad esse non sono associati gli eventi *Click* e *DblClick*).
Si disegnano due rettangoli anziché utilizzare Frame (sono essi stessi dei controlli e quindi ciascuno avrebbe un proprio evento *DragDrop*).

```
Private Sub Form_Load ()
    CurrentY = ScaleWidth / 20
    CurrentX = ScaleWidth / 9
    Print "Ufficio";
    CurrentX = 5 * ScaleWidth / 9
    Print "Casa"
    Line (ScaleWidth/9, 2*ScaleHeight / 9)-(4 * ScaleWidth / 9, 8 * ScaleHeight / 9), , B
    Line (5 * ScaleWidth/9, 2*ScaleHeight / 9)-(8*ScaleWidth/9, 8*ScaleHeight / 9), , B
End Sub
```

Ora bisogna inserire le Label contenenti i nomi delle applicazioni e rendere possibile il loro trascinamento.
Si creino quattro Label con Caption = LLL per creare una matrice di controlli.

Il metodo Drag

Control.Drag	0	annulla il trascinamento.
Control.Drag	1	lo inizia.
Control.Drag	2	lo termina.

Il metodo *Drag* è necessario solo se *DragMode* è impostato a manuale (0), ma può essere utilizzato anche se è impostato ad automatico (1).

L'evento DragDrop: si verifica quando un controllo è rilasciato su un oggetto.
L'evento DragOver: si verifica quando un controllo è trascinato su un oggetto.
La proprietà DragIcon: indica l'icona da utilizzare quando l'utente trascina un controllo.

La proprietà DragMode
Può essere impostata a manuale (0, default), o automatico (1) per controllare il trascinamento dei controlli.
Se è automatico, l'utente può trascinarlo quando lo desidera ma il controllo non risponde ad altre azioni del mouse.
Se è manuale il trascinamento deve essere gestito con il metodo *Drag*.
Si utilizza il trascinamento manuale, quando l'utente preme un pulsante mentre il puntatore del mouse si trova su un controllo il cui *DragMode* è impostato a manual, è generato un evento *MouseDown()* con il seguente codice.

```
Private Sub LLL_MouseDown (Index As Integer, Button As Integer, Shift As Integer,
X As Single,Y As Single)
    LLL(Index).Visible = 0          'rende invisibile la Label per trascinarla
    LLL(Index).Drag 1
End Sub
```

L'operazione successiva consiste nel rispondere quando l'utente rilascia il pulsante del mouse, posizionando la Label; l'evento che si verifica è *DragDrop* e gli argomenti passati sono le coordinate x e y della nuova posizione ed il controllo che è stato spostato.

```
Private Sub Form_DragDrop (Source As Control, X As Single, Y As Single)
    If TypeOf Source Is Label Then
        Source.Drag 2
        Source.Visible = -1          'rende visibile la Label
        Source.Move (X - Source.Width / 2), (Y - Source.Height / 2)
    End If
End Sub
```

Si può utilizzare Source come un qualunque controllo, ma come si fa a sapere che tipo di controllo è stato spostato?
I nomi Label1 o Text1 non sono disponibili durante l'esecuzione, allora si può conoscere il tipo di controllo con *TypeOf* (al posto del segno di uguaglianza si usa *Is*).
Visual BASIC non sposta la Label che è stata trascinata nella nuova posizione nel momento in cui è rilasciata, ma è il programmatore con il metodo *Move* che sposta il controllo nella nuova posizione. Adesso bisogna rendere attiva la procedura *DblClick().*

```
Private Sub LLL_DblClick (Index As Integer)
    Dim Result As Integer
    Resultval = Shell(LLL(Index).Tag, 1)
End Sub
```

Quale applicazione è avviata?
Si usa la proprietà *Tag* che consente di associare una stringa ad un controllo, in modo che sia possibile durante l'esecuzione identificare il controllo su cui si sta lavorando, in questo caso si immagazzineranno il percorso ed il nome del file di ciascuna applicazione nella proprietà *Tag* della Label corrispondente.

SPREAD.VBP

Il controllo Grid consente di visualizzare le informazioni in celle. Le righe e le colonne possono essere di tipo:
- ✓ fisso, non è possibile eseguire scorrimenti (colore grigio);
- ✓ mobile, consentono di eseguire scorrimenti (colore bianco).
- ✓ Non è possibile inserire testo in fase di progettazione, ma solo in fase di esecuzione con le proprietà:
- ✓ Text, inserisce testo in una cella o lo stesso testo in un intervallo;

- ✓ Clip, inserisce testo diverso in più celle, è necessario usare TAB Chr (9) per indicare la cella successiva e CR Chr (13) per indicare la riga successiva.

Oggetto	Proprietà	Impostazione
Form1	Caption	Spreadsheet
	Windowstate	Maximized
Grid	Cols	15
	Rows	25
	ColWidth	larghezza colonna
	RowHeight	altezza riga

Si assegnano le etichette alle celle del foglio, nell'evento *Form_Load(),* i numeri vanno alle righe e le lettere alle colonne.

```
Private Sub Form_Load()
    Static Items(1 To 6) As String
    Dim loop_index As Integer
    Grid1.ColWidth(2) = 1000
    Grid1.ColWidth(1) = 1000
    Items(1) = "Affitto"
    Items(2) = "Cibo"
    Items(3) = "Automobile"
    Items(4) = "Telefono"
    Items(5) = "Gas"
    tems(6) = "TOTALE"
    Grid1.Row = 1
    For loop_index = 1 To 24
        Grid1.Col = 0
        Grid1.Row = loop_index
        Grid1.Text = Str$(loop_index)
    Next loop_index
    For loop_index = 1 To 6
        Grid1.Col = 0
        Grid1.Row = loop_index
        Grid1.Text = Str$(loop_index)
        Grid1.Col = 2
        Grid1.Text = Items(loop_index)
    Next loop_index
    Grid1.Col = 1
    Grid1.Row = 0
    For loop_index = 1 To 14
        Grid1.Col = loop_index
        Grid1.Text = Chr$(Asc("A") - 1 + loop_index)
    Next loop_index
    ' si evidenzia la cella corrente
    Grid1.Row = 1
    Grid1.Col = 1
End Sub
```

Ora, si possono leggere i valori inseriti nelle celle (tutte quelle a cui si è assegnata un'etichetta, eccetto l'ultima), a questo scopo si usa l'evento:

```
Private Sub Grid1_KeyPress(KeyAscii As Integer)
    Dim OldRow, OldCol, row_index As Integer
    Dim sum As Currency
    Grid1.Text = Grid1.Text + Chr$(KeyAscii)
    OldRow = Grid1.Row
    OldCol = Grid1.Col
    Grid1.Col = 1       'Somma i numeri nella prima colonna
```

```
        Grid1.Row = 0
        sum = 0
        For row_index = 1 To 5
            Grid1.Row = row_index
            sum = sum + Val(Grid1.Text)
        Next row_index
        Grid1.Row = 6   'si inserisce il valore di sum% nella cella del totale
        Grid1.Text = Format(Str$(sum), "Currency")
        Grid1.Row = OldRow
        Grid1.Col = OldCol
    End Sub
```

ALARM.VBP

Il controllo Timer è collegato al trascorrere del tempo, per esempio il controllo dell'orologio del sistema.
La proprietà *Interval* (valore compreso tra 1 e 65.535) non determina la durata, ma la frequenza.
Maggiore è la frequenza (valori piccoli di *Interval*), maggiore è il tempo di elaborazione utilizzato per la risposta agli eventi, con conseguente rallentamento delle prestazioni generali del sistema.
Per tenere conto di un certo margine di errore, si consiglia di impostare un intervallo pari a metà del livello di precisione desiderato.
Per esempio, 500 per 1 secondo. Inserire nel form.

Oggetto	Proprietà	Impostazione
TextBox	Name	AlarmSettingl
Label	Name	Display
Timer	Interval	1000 (1000 msec = 1 sec)

```
    Private Sub Display_Click ()
        Display.Caption = Time$
    End Sub
```

Ogni volta che l'utente digita, si verificano tre eventi:
1. KeyDown preme un tasto;
2. KeyUp rilascia un tasto;
3. KeyPress ritorna il valore del carattere.

Ogni routine di evento accetta due argomenti:
1. *Keycode* codice ANSI (non ASCII) del tasto premuto o rilasciato;
2. *Shift* riporta lo stato dei tasti MAIUSC, ALT e CTRL.

Inoltre, KeyDown e KeyUp non considerano la differenza tra maiuscolo e minuscolo, allora bisogna esaminare *Shift*: questo argomento è uno se MAIUSC è stato premuto, due se è premuto CTRL, quattro per ALT.
Per esempio, vale cinque se sono premuti contemporaneamente ALT e MAIUSC.
Lo scopo è quello di limitare la digitazione dell'utente, nell'esempio i caratteri da zero a nove ed i due punti; doppio clic su AlarmSetting e selezionare KeyPress, il codice è:

```
    Private Sub AlarmSetting_KeyPress (KeyAscii As Integer)
        Key$ = Chr$(KeyAscii)
        If ((Key$ < "0" Or Key$ > "9") And Key$ <> ":") Then
            Beep
            KeyAscii = 0
        End If
    End Sub
```

Il codice ASCII in *KeyAscii* è convertito in carattere e confrontato, se non è corretto emette un *Beep* ed è cancellato.
Il **Timer** si usa per mantenere aggiornato l'orologio, produce l'evento Timer, il codice è il

seguente:

```
Private Sub Timer1_Timer ()
    If (Time$ >= AlarmSetting.Text And AlarmOn) Then Beep
    Display.Caption = Time$
End Sub
```

All'orario impostato per la sveglia, questa routine emette un Beep.
Gli orologi però hanno due possibilità: allarme attivo o inattivo, per questo motivo bisogna aggiungere alla destra di Textbox due Optionbutton: Alarm On e Alarm Off. La routine collegata a questi pulsanti deve poter comunicare con quella corrente, *Timer1_Timer()*, tramite una variabile globale: *Private AlarmOn As Integer*.
L'ultima istruzione aggiorna il display. Alarm Off deve essere attivo quando l'utente avvia l'orologio.
Sono pulsanti di scelta mutuamente esclusivi, il modo più semplice per gestirli è quello di assegnare lo stesso Name = OnOffButton, Visual BASIC assegna automaticamente un indice e chiede se deve creare l'array, doppio clic su uno dei due pulsanti per il codice:

```
Private Sub OnOffButton_Click (Index As Integer)
    If (Index = 1) Then
        AlarmOn = True
    Else
        AlarmOn = False:End If
End Sub
```

ALARM1.VBP

Cancellare i due pulsanti; la proprietà Caption della singola voce di menu permette di modificare la voce stessa durante l'esecuzione.
Disegnare il menu.

Caption	Name
Alarm	AlarmMenu
AlarmOff	OnOffItem

Clic sulla voce di menu.

```
Private Sub OnOffItem_Click ()
    If (AlarmOn) Then
        AlarmOn = False
        OnOffItem.Caption = "Alarm Off"
    Else
        AlarmOn = True
        OnOffItem.Caption = "Alarm On"
        OnOffItem.Checked = True
    End If
End Sub
```

ALARM2.VBP

Disegnare il menu.

Caption	Name
Alarm	AlarmMenu
AlarmOn	AlarmOnOnItem
AlarmOff	AlarmOffItem

Clic su AlarmOn per il codice.

```
Private Sub AlarmOnItem_Click ()
    If (AlarmOn) Then
        AlarmOn = False
        AlarmOffItem.Visible = True
        AlarmOnItem.Visible = False
```

```
            AlarmOffItem.Checked = True
        Else
            AlarmOn = True
            AlarmOnItem.Visible = True
            AlarmOffItem.Visible = False
            AlarmOnItem.Checked = True
        End If
    End Sub
```

CONVERSIONI.VBP

Il Command1 esegue la conversione binario-decimale.

```
    Private Sub Command1_Click()
        Dim r, a, i, x As Integer
        Dim RetString As String
        RetString = InputBox("Inserisci il numero binario - non esistono controlli! - partendo dall'MSB...", "Numeri Binari")
        If RetString = "" Then Exit Sub
        r = 0
        a = Len(RetString$)
        For i = 1 To Len(RetString$)
            a = a - 1
            x = Val(Mid$(RetString$, i, 1))
            r = (r + (x * 2 ^ a))
        Next i
        Text1.Text = "Il numero decimale vale: " & r
    End Sub
```

Il Command2 esegue la conversione decimale-binario.

```
    Private Sub Command2_Click()
        Dim r(1 To 32) As Integer
        Dim x, n, quoz, resto, i As Integer
        Dim RetString As String
        Form1.Cls
        Text1.Text = ""
        RetString = InputBox(" Inserisci  il  numero - non esistono controlli! - decimale...", "Numeri Decimali")
        If RetString = "" Then Exit Sub
        x = 1
        n = Val(RetString$)
        a = n
        If n <> 1 Then
            Do
                'quoz = Int(n / 2)
                quoz = n \ 2
                'resto = (n - (2 * (Int(n / 2))))
                resto = n Mod 2
                n = quoz
                r(x) = resto
                x = x + 1
            Loop Until quoz = 1
        End If
        r(x) = 1
        Print: Print: Print: Print: Print: Print: Print: Print
        Print "IL NUMERO DECIMALE "; a; " IN BINARIO DIVENTA ": Print
```

```
        For i = x To 1 Step -1
            Print r(i);
            'Text1.Text = r(i)
        Next i
    End Sub
```

QUICKSORT.VBP

```
    Private Array(1 To 9) As Integer
```

Il Command1 esegue la ricerca binaria.

```
    Private Sub Command1_Click()
        Dim basso, alto, meta, SearchValue As Integer
        Array(1) = 1
        Array(2) = 2
        Array(3) = 3
        Array(4) = 4
        Array(5) = 5
        Array(6) = 6
        Array(7) = 7
        Array(8) = 8
        Array(9) = 9
        SearchValue = 8
        Print
        Print "Ricerca del valore 8 nell'elenco ordinato"
        Print
        basso = LBound(Array, 1)
        alto = (UBound(Array, 1) - 1)
        Do While (basso <= alto)
            meta = (basso + alto) / 2
 I          if SearchValue = Array(meta) Then
                Print "Valore"; SearchValue; "nell'elenco"
                Exit Do
            ElseIf SearchValue < Array(meta) Then
                alto = meta - 1
            Else: basso = meta + 1
            End If
        Loop
    End Sub
```

Il Command2 esegue l’ordinamento con il metodo del quick sort.

```
    Private Sub Command2_Click()
        Dim i As Integer
        Array(1) = 9
        Array(2) = 8
        Array(3) = 7
        Array(4) = 6
        Array(5) = 5
        Array(6) = 4
        Array(7) = 3
        Array(8) = 2
        Array(9) = 1
        Print
        Print " i           Array(i)"
        Print "---           --------"
        For i = 1 To 9
```

```
                Print i, Array(i)
            Next i
            Call SortQuick(Array(), 1, UBound(Array, 1))
            Print
            Print "Ordinamento..."
            Print
            Print " i              Array(i)"
            Print "---             --------"
            For i = 1 To 9
                Print i, Array(i)
            Next i
        End Sub
Private Sub SortQuick(ByRef Array() As Integer, ByVal SortFrom As Integer, ByVal SortTo
As Integer)
        Dim Temp, i, j As Integer
        If SortFrom >= SortTo Then Exit Sub
        If SortFrom + 1 = SortTo Then  'Ultimo caso
            If Array(SortFrom) > Array(SortTo) Then
                Temp = Array(SortFrom)
                Array(SortFrom) = Array(SortTo)
                Array(SortTo) = Temp
            End If
        Else   'Il problema va suddiviso
            AtRandom = (SortFrom + SortTo) \ 2
            Test = Array(AtRandom)
            Temp = Array(AtRandom)
            Array(AtRandom) = Array(SortTo)
            Array(SortTo) = Temp
            Do  'Divisione in due partizioni
                For i = SortFrom To SortTo - 1
                    If Array(i) > Test Then Exit For
                Next i
                For j = SortTo To i + 1 Step -1
                    If Array(j) < Test Then Exit For
                Next j
                If i < j Then
                    Temp = Array(i)
                    Array(i) = Array(j)
                    Array(j) = Temp
                End If
            Loop Until i >= j
            Temp = Array(i)
            Array(i) = Array(SortTo)
            Array(SortTo) = Temp
            Call SortQuick(Array(), SortFrom, i - 1)
            Call SortQuick(Array(), i + 1, SortTo)
        End If
End Sub
```

GRAFICA

INTRODUZIONE

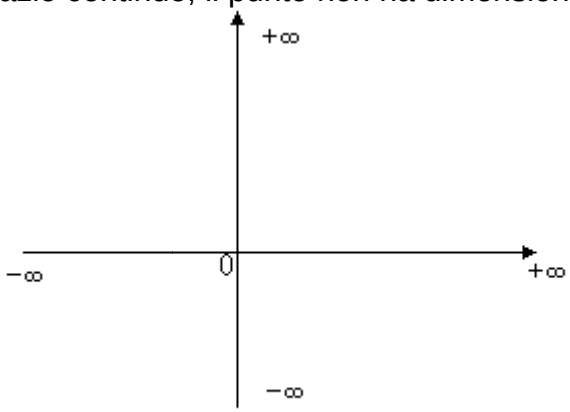
È possibile disegnare in due tipi di oggetti: i form e le PictureBox.
In entrambi si deve impostare la proprietà *Autoredraw* = True, per permettere a Visual BASIC di ridisegnare automaticamente la grafica presente nel caso sia coperta da altre finestre.
Si possono usare:

- ✓ controlli (oggetti) grafici: Image, Line, Shape;
- ✓ metodi grafici: *Cls*, *PSet*, *Line*, *Circle*.

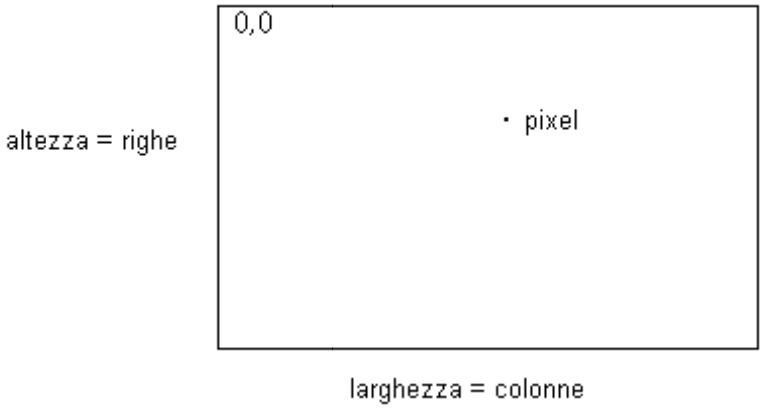
Esistono cinque proprietà predefinite:

1. *Height* altezza esterna della finestra incluse la barra del titolo e del menu;
2. *Width* larghezza esterna della finestra;
3. *ScaleWidth* altezza dell'area di disegno in twip;
4. *ScaleHeight* larghezza dell'area di disegno;
5. *DrawWidth* dimensione dell'oggetto grafico, per default un pixel.

Spazio geometrico
Il piano cartesiano è uno spazio continuo, il punto non ha dimensioni.



Monitor: spazio discreto
Il sistema di coordinate parte da (0,0), angolo superiore sinistro.



Disegno di punti

[oggetto.]PSet [Step] (x,y) [,Color&]

dove l'argomento *Step* indica che le coordinate sono relative alla posizione corrente, specificata dalle proprietà *Currentx* e *Currenty*.

PSet (ScaleWidth / 2, Scaleheight / 2), RGB(255, 0, 0) 'pixel al centro della form
Picture1.PSet (Picture1.ScaleWidth / 2, Picture1.Scaleheight / 2), RGB(255, 0, 0)

Cancellare il punto
Colorare il corrispondente pixel con il colore dello sfondo: *Oggetto.Pset (x,y), BackColor*

Cancellare l'area grafica
Con il colore di sfondo: *oggetto.Cls*

Ottenere il colore del punto
oggetto.Point (x,y)

Definire lo spessore della penna
Oggetto.DrawWidth = valoreintero

Disegno di linee
Esistono due possibilità:
1. lo strumento Line, usato però per impostare l'aspetto grafico delle finestre;
2. *[oggetto.]Line [[Step] (x1,y1)] - [Step] (x2,y2) [, [color&], B [F]]*

For i = 1 To 9
DrawWidth = i
*Line (0, i * Scaleheight / 10)-(ScaleWidth, i * Scaleheight / 10)*
Next i
DrawWidth = 1

Se non si specifica il punto di origine, è utilizzata la posizione grafica corrente, impostata al momento della pressione iniziale del pulsante del mouse: utile per disegnare a mano libera *Line - (x,y).* Sono disponibili sette diversi stili di linea con la proprietà *DrawStyle*:

0 Linea continua, valore di default
1 Linea tratteggiata
2 Linea punteggiata
3 Linea a trattini e punti
4 Linea a trattini e punti, due punti ogni trattino
5 Linea trasparente, non è visualizzata
6 Linea interna

For i = 1 To 7
DrawStyle = i - 1
*Line (0, i * Scaleheight / 8)-(ScaleWidth, i * Scaleheight / 8)*
Next i

Per *DrawWidth* maggiore di uno, gli stili di linea da uno a quattro risultano linee continue.
Sono disponibili diversi tipi di penna con la proprietà *DrawMode*:

4 *Not Pen* Disegna nel colore inverso a quello della penna
6 *Invert Pen* Inverte il contenuto dello schermo
7 *Xor Pen* Effettua uno xor di penna e schermo
11 *No Pen* Non disegna nulla
13 *Copy Pen* Disegna direttamente col colore della penna, default

DrawMode = 6
DrawWidth = 9
Line (0, 0)-(ScaleWidth, Scaleheight)
Line (0, Scaleheight)-(ScaleWidth, 0)

Quando si effettua uno XOR rispetto a tutti uno, si ottiene il suo inverso: il bianco, per esempio, diventa nero.
Se si effettua poi un nuovo xor si riotterrà il valore originale, bianco.

Questo significa che si può disegnare usando la *xor Pen* per far apparire qualcosa, e ridisegnando una seconda volta con la stessa penna il disegno precedente scompare: in questo modo funziona l'animazione e la crittografia.

Disegno di rettangoli

Esistono due possibilità:

1. lo strumento Shape, usato però per impostare l'aspetto grafico delle finestre;
2. opzione *B* del metodo *Line*

```
DrawWidth = 8
DrawStyle = 6
DrawMode = 6
Line (0, 0)-(ScaleWidth / 2, Scaleheight / 2), , B
Line (ScaleWidth / 4, Scaleheight / 4)-(3 * ScaleWidth / 4, 3 * Scaleheight / 4), , B
Line (ScaleWidth / 2, Scaleheight / 2)-(ScaleWidth, Scaleheight), , B
DrawWidth = 1
```

Sono disponibili diversi motivi di riempimento con la proprietà *FillStyle*

0	Uniforme
1	Trasparente, default
2	Linee orizzontali
3	Linee verticali
4	Diagonali verso l'alto
5	Diagonali verso il basso
6	Intreccio
7	Intreccio diagonale

L'opzione *F* del metodo *Line* è usata per specificare che il retino di riempimento deve essere dello stesso colore del rettangolo.

Il disegno di un rettangolo bianco uniforme, con bordi bianchi, è il metodo standard per cancellare il testo all'interno di form e PictureBox.

```
SX = ScaleWidth
SY = ScaleHeight
For i = 0 To 3
    FillStyle = i
    Line ((2 * i + 1) * SX / 9, SY / 5) - ((2 * i + 2) * SX / 9, 2 * SY / 5), , B
    FilleStyle = i + 4
    Line ((2 * i + 1) * SX / 9, 3 * SY / 5) - ((2 * i + 2) * SX / 9, 4 * SY / 5), , B
Next i
```

Disegno di cerchi

Si usa il metodo:

```
[oggetto.]Circle [Step] (x,y), raggio [, [color&] [, [inizio] [, [fine] [, rapporto]]]]
FillStyle = 5
ForeColor = RGB (255, 0, 0)
FillColor = ForeColor                    'assegna al retino lo stesso colore
Circle (ScaleWidth / 4, Scaleheight / 4), Scaleheight / 5
FillStyle = 2
Circle (ScaleWidth / 2, Scaleheight / 2), Scaleheight / 3, , , , 2        'ellisse
Circle (3 * ScaleWidth / 4, 3 * Scaleheight / 4), Scaleheight / 5, , 0, 3.1415 'arco
```

Modifica del sistema di coordinate

Si usa il metodo: *Scale (xalto, ysinistra) - (xdestra, ybasso)*

Con questo comando, le coordinate dell'angolo superiore sinistro sono impostate a (*xalto*, *ysinistra*) e quelle dell'angolo inferiore destro a (*xdestra*, *ybasso*).

Si apra il progetto **plotter.vbp** con il seguente codice.

```
Private XVal(4), YVal(4) As Single      'variabili a livello form
```

```
    Private XBiggest, YBiggest As Single
```
Questo evento si verifica ogni volta che le dimensioni del form sono modificate.
```
    Private Sub Form_Resize ()
        Call Form_Paint
    End Sub
    Private Sub Form_Paint ()          'programma grafico, assume l'origine a (0, 0)
        Dim loop_index As Integer
        Cls
        XVal(1) = 1
        YVal(1) = 200
        XVal(2) = 2
        YVal(2) = 95
        XVal(3) = 4
        YVal(3) = 350
        XVal(4) = 5
        YVal(4) = 425
        XBiggest = 0
        For loop_index = 1 To UBound(XVal)
            If XVal(loop_index)>XBiggest Then XBiggest = XVal(loop_index)
        Next loop_index
        YBiggest = 0
        For loop_index = 1 To UBound(YVal)
            If YVal(loop_index)>YBiggest Then YBiggest = YVal(loop_index)
        Next loop_index
        Scale (0, YBiggest)-(XBiggest, 0)          'assume (0, 0) come origine
        CurrentX = XVal(1)
        CurrentY = YVal(1)
        For loop_index = 1 To UBound(XVal)
            Line -(XVal(loop_index), YVal(loop_index))
        Next loop_index
    End Sub
```
Il metodo *Form_Resize()* permette di ricreare l'immagine dopo che la finestra è stata ridotta ad icona o coperta da un'altra applicazione.
Il modo più efficace è mettere tutto il codice per la grafica all'interno di *Form_Paint()*. Un'alternativa, senza usare codice, è utilizzare la proprietà *Autoredraw* = True con un enorme impegno di memoria.

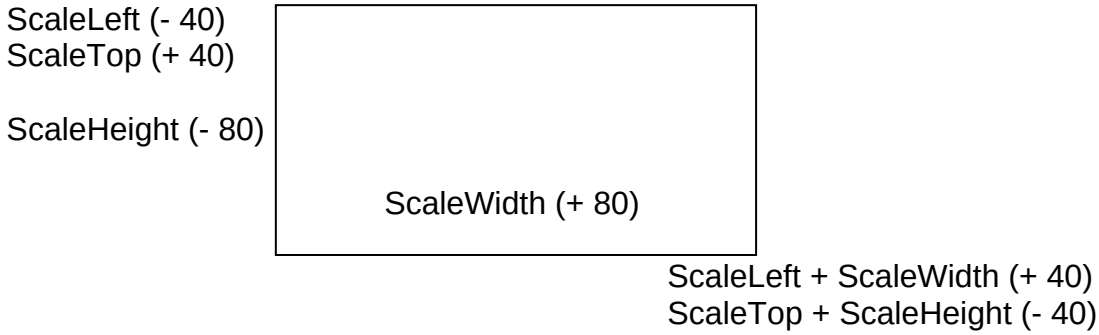
La scala nella grafica
L'unità di misura di default è il twip, a volte però è più comodo usarne altre, per esempio i pixel. A tal fine, si usi la proprietà *ScaleMode*:

0 Definita dall'utente, *ScaleWidth* e *ScaleHeight* già impostate
1 Twip
2 Punti (72 punti per pollice)
3 Pixel (*ScreenWidth* e *ScreenHeight* dimensione dello schermo in pixel)
4 Carattere (120 twip sull'asse delle ascisse, 240 twip sull'asse delle ordinate)
5 Pollici (1440 twip)
6 Millimetri
7 Centimetri (8567 twip)

```
Private Sub Command1_Click()
   Dim i As Integer
   For i = 0 To 7
      Me.ScaleMode = i
        Text1.Text = Text1.Text & "Con ScaleMode =" & Me.ScaleMode & " ScaleWidth
vale " & Me.ScaleWidth & " e ScaleHeight vale " & Me.ScaleHeight & vbCrLf
```

```
    Next i
    Text1.Text = Text1.Text & "Width vale sempre " & Me.Width & "e Height " & Me.Height
End Sub
```

Non è necessario che le unità di altezza siano in relazione con quelle di larghezza, si può fissare come origine delle coordinate un punto diverso dall'angolo superiore sinistro con *ScaleLeft* e *ScaleTop*.
Le coordinate al centro dello schermo (0,0) *Scale* (-40,40) - (40,-40) sono:




```
Private Sub Form_Load()
    Scale (-40, 40)-(40, -40)
End Sub
Private Sub Command1_Click()
    Line (-40, 0)-(40, 0)
    Line (0, 40)-(0, -40)
End Sub
```

Caricamento d'immagine
È possibile inserire immagini ICO, BMP e WMF in:
- ✓ form, con la proprietà *Picture* oppure con *[oggetto.]Picture=LoadPicture (Filename$)*
- ✓ PictureBox, con la proprietà *Picture* e *AutoSize* = True
- ✓ Image, con la proprietà *Picture*; non dispone dei metodi grafici standard come *Line* e *Circle* e differisce da una PictureBox per due motivi: le sue dimensioni sono impostate automaticamente in modo da corrispondere a quelle dell'immagine caricata e ponendo *Stretch* = True se ne possono modificare le dimensioni anche dopo il caricamento dell'immagine.

Le immagini, così come possono essere caricate, possono essere salvate con:

```
    SavePicture [oggetto.]Image, Filename$
```

Per implementare lo scorrimento verticale e orizzontale di un bitmap di dimensioni maggiori della finestra, si utilizza la coppia di PictureBox: Picture1 (*Autosize* = False) contenitore vuoto e Picture2 (*Autosize* = True) che contiene effettivamente l'immagine disegnata all'interno di Picture1.

```
    Private Sub Form_Load ()
        Picture2.Move 0, 0
        HScroll1.Max = Picture2.Width - Picture1.Width
        VScroll1.Max = Picture2.Height - Picture1.Height
    End Sub
    Private Sub HScroll1_Change ()
        Picture2.Left = -HScroll1.Value
    End Sub
    Private Sub VScroll1_Change ()
        Picture2.Top = -VScroll1.Value
    End Sub
```

Copiare un'immagine da un oggetto all'altro: *Set Picture1.Picture = Picture2.Picture*
Rimuovere un'immagine: *Set Picture1.Picture = LoadPicture ("")*
La proprietà *Left* indica la distanza tra l'angolo superiore sinistro del controllo e il lato

sinistro del form. La proprietà *Top* indica la distanza tra l'angolo superiore sinistro del controllo e la parte inferiore del form.

Animazioni

Visual BASIC non gestisce la manipolazione dei pixel ad una velocità sufficiente per poter realizzare vere e proprie animazioni.

Per esempio, nel progetto **sprite.vbp** è necessario impostare *ScaleMode* = Pixel e *BackColor* = Black. Inserire il seguente codice in ComamndButton.

```
Private Sub Command1_Click ()
    Static SpriteArray(1 To 16) As String
    Static Sprite(16, 16) As Long
    Dim x, y, StepSize, i, ii, j, jj, CurrentColor As Integer
    SpriteArray(1) = "00000000E0000000"
    SpriteArray(2) = "00000000E0000000"
    SpriteArray(3) = "0000000EEE000000"
    SpriteArray(4) = "000000EE3EE00000"
    SpriteArray(5) = "000000EE3EE00000"
    SpriteArray(6) = "00000EE333EE0000"
    SpriteArray(7) = "00000EE333EE0000"
    SpriteArray(8) = "00E00EE333EE00E0"
    SpriteArray(9) = "00EE00EE3EE00EE0"
    SpriteArray(10) = "00EEE0EE3EE0EEE0"
    SpriteArray(11) = "00EEEEEE3EEEEEE0"
    SpriteArray(12) = "00EEE00E3E00EEE0"
    SpriteArray(13) = "00EE004444400EE0"
    SpriteArray(14) = "00E00004440000E0"
    SpriteArray(15) = "0000000444000000"
    SpriteArray(16) = "0000000040000000"
    DrawMode = 7
    BackColor = 0
    GoSub MakeSprite
    x = Int(ScaleWidth / 3)
    y = ScaleHeight - UBound(SpriteArray, 1)
    GoSub DrawSprite
    Stepsize = Int(ScaleHeight / 10)
    For i = (ScaleHeight - UBound(Sprite, 1)) To Stepsize Step -Stepsize
        y = i
        GoSub DrawSprite
        y = i - Stepsize
        GoSub DrawSprite
    Next i
    Exit Sub
MakeSprite:
    For i = 1 To UBound(SpriteArray, 1)
        For j = 1 To Len(SpriteArray(i))
            CurrentColor = Asc(UCase$(Mid$(SpriteArray(i), j, 1)))
            If CurrentColor<=Asc("9") Then
                CurrentColor=CurrentColor-Asc("0")
            Else
                CurrentColor = CurrentColor - Asc("A") + 10
            End If
            Sprite(j, i) = QBColor(CurrentColor)
        Next j
    Next i
```

```
    Return
    DrawSprite:
        For ii = 1 To UBound(Sprite, 1)
            For jj = 1 To UBound(Sprite, 2)
                PSet (x + ii - 1, y + jj - 1), Sprite(ii, jj)
            Next jj
        Next ii
    Return
    End Sub
```

L'ideale per le animazioni è realizzare l'immagine bitmap, quindi caricarla in Image tramite la proprietà *Picture*.
Nella terminologia tecnica, l'immagine è definita sprite, dato che si tratta di un bitmap che non è modificato, ma solo mosso sullo schermo.
Aprire il progetto **sprite1.vbp**.

Oggetto	Proprietà	Impostazione
AniPushButton	Frame	inserire i file moon01-08.ico

```
    Private Sub AniButton1_Click ()
        Dim loop_index As Integer
        For loop_index = Image1.Top To 0 Step -50
            Image1.Move Image1.Left, loop_index
        Next loop_index
    End Sub
```

Per associare l'icona dell'astronave all'applicazione si usa la proprietà del form di nome *Form1.Icon* = sprite1.ico.

Controlli grafici

Il controllo grafico è Graph, i dati si caricano nel modo seguente.

Oggetto	Proprietà	Impostazione
Graph	Graphdata	200 <CR>95<CR>0<CR>350<CR>425<CR>.
	ColorData	(0..15),
	GraphType	
	PatternData	(1..31)
	GridStyle	
	GraphTitle	
	LeftTitle	
	BottomTitle	

Si noti che non si è incrementato un contatore per puntare al successivo valore x, inoltre imposta automaticamente la scala delle y e non si è inserito codice.
Si può modificare il colore, i motivi di riempimento, il grafico, aggiungere una griglia, dei titoli.
Per cambiare il valore di uno solo dei punti si utilizza la proprietà *ThisPoint*, per esempio con il valore quattro salta il punto tre.
Il grafico è spostato di una posizione lungo l'asse delle x; per correggere questo problema, si può impostare la proprietà *XposData*. Da un grafico a torta per estrarre una fetta si usa *ExtraData* = 1

Visualizzazione del testo grafico

Può essere stampato in tre contesti:

1. nel form
2. nella PictureBox
3. sulla stampante

Si usa il metodo: *[oggetto.]Print [listaespressioni] [; | ,]*
dove con il ';', il cursore è posizionato al termine del testo che si stampa; mentre con la ',' il cursore è posizionato nella zona (TAB, 14 caratteri) di stampa successiva.

```
Private Sub Form_Load ()
    Print:Print:Print
    Print Tab(10); "REGIONE"; Tab(30); "PRODOTTO"
    Print
    Print Tab(10); "Sicilia"; Tab(30); "Arance"
    Print Tab(10); "Piemonte"; Tab(30); "Uva"
    Print Tab(10); "Lombardia"; Tab(30); "Patate"
    Print Tab(10); "Toscana"; Tab(30); "verze"
    DrawWidth = 2
    Line (650, 400)-(3700, 1900), , B:Line (650, 930)-(3700, 930)
End Sub
```

Il testo è stampato dalla posizione corrente del cursore, per modificarla si usa come al solito *Currentx* e *Currenty*.
Il metodo *Cls* elimina il contenuto del form.
Se si stampa in una PictureBox si deve conoscere la lunghezza della stringa, altrimenti il testo è troncato a destra. Si usano due metodi:

1. *StringLength% = Picture1.TextWidth ("...")* *'lunghezza della stringa*
2. *Height% = TextHeight (2...")* *'altezza di una riga*

L'esempio relativo si chiama **stampa.vbp**, da notare che nel codice c'è un altro particolare importante: **il passaggio diretto di controlli come argomenti alle routine**.
È consigliabile in due casi:

1. quando si hanno numerosi controlli sui quali si devono svolgere operazioni ripetute, come nel caso in cui debbano essere inizializzati;
2. quando non si sa su quale controllo si dovrà agire, come in stampa.vbp.

Il codice relativo è il seguente.

```
Private StringToPrint As String
Sub Form_Load ()
    Call PrintString(Picture1, "Questo è il momento per tutti gli uomini...")
End Sub
```

All'interno di *PrintString()* ci si può riferire alla PictureBox semplicemente come *PBox*.

```
Private Sub PrintString (PBox As Control, PString As String)
'la formattazione del testo in una Picture di dimensioni limitate costituisce un problema 'frequente. Si suddivide la stringa da stampare PString in righe che possano essere 'contenute nella Picture.
    Dim PrintLine, NextWord, Temp As String
    StringToPrint = PString:PrintLine = ""
'GetWords restituisce la parola successiva in PString
    NextWord = GetWord()
    Do While NextWord <> ""              'termina quando c'è una stringa nulla
        Temp = PrintLine + NextWord
        If PBox.TextWidth(Temp) > PBox.ScaleWidth Then
            PBox.Print PrintLine  'Stampa prima che diventi troppo lunga.
            PrintLine = NextWord
        Else
            PrintLine = Temp
        End If
        NextWord = GetWord()
    Loop
    PBox.Print PrintLine                 'Stampa il rimanente.
End Sub
Function GetWord () As String
'lo scopo è quello di tagliare la prima parola della stringa StringToPrint, in altre parole si 'deve trovare il primo spazio nella stringa e restituire tutto ciò che lo precede, nonché lo spazio stesso si usa la funzione Instr() per determinare se in StringToPrint c'è uno uno
```

```
'spazio, se è uguale a zero non vi sono spazi.
    If InStr(StringToPrint, " ") = 0 Then
        GetWord = StringToPrint
        StringToPrint = ""
'se c'è, significa la stringa si compone almeno di due parole si deve restituire la prima e tagliarla dalla stringa in modo da preparala per la successiva chiamata di GetWords$
        Else
            GetWord = Left$(StringToPrint, InStr(StringToPrint, " "))
    StringToPrint=Right$(StringToPrint,Len(StringToPrint)-InStr(StringToPrint, " "))
        End If
End Function
```

Esistono delle proprietà dei font su cui è possibile intervenire quando si stampa a video o su stampante, inclusi gli attributi del testo:

FontName	Nome del font
FontSize	Corpo del font in punti
FontBold	Stampa in grassetto se true
FontItalic	Stampa in corsivo se true
FontStrikethru	Stampa in barrato se true
FontTransparent	Determina se è utilizzato lo sfondo intorno ai caratteri
FontUnderline	Stampa in sottolineato se true

Uso della stampante

Si usa il metodo:

```
Private Sub Form_Load ()
    Form1.PrintForm
End Sub
```

che consente si stampare un'intero form, tuttavia, in questo modo si produce una stampa bitmap (pixel per pixel del contenuto dello schermo).

Per ottenere una risoluzione più alta, si deve usare il metodo *Print* applicato all'oggetto *Printer*, che corrisponde alla stampante di default di Windows.

```
Private Sub Form_Load ()
    Printer.CurrentX = 1440              'posiziona l'output di stampa
    Printer.CurrentY = 2880
    Printer.FontName = "Courier"
    Printer.FontUnderline = True
    Printer.Print "..."
    Printer.NewPage
End Sub
```

FILE

INTRODUZIONE

Le applicazioni devono essere in grado di memorizzare delle informazioni su dei supporti di memoria di massa in modo da renderle disponibili per elaborazioni successive o renderle trasportabili ad altri sistemi o utenti.
Le informazioni non devono essere necessariamente omogenee, in altre parole, dello stesso tipo; possiamo scrivere informazioni strutturate, in pratica costituite da insiemi disomogenei di dati eventualmemte correlati tra di loro, oppure informazioni di natura totalmente diversificata.

FILE SEQUENZIALI

Costituiscono la struttura file più elementare (a righe) e sono utilizzati per memorizzare un'informazione non strutturata, spesso in forma testuale: file di testo.
Per ricercare un'informazione bisogna scorrerlo (leggerlo) fino al punto che c'interessa: è evidente che se la dimensione è grande si allungano i tempi di accesso.
Si aggiunge il supporto dei file all'applicazione **editor1.vbp**. Si inizia dalla voce Apri...

1. Selezionare la Common dialog nella ToolBox.
2. Posizionarla nel form, durante l'esecuzione non è visibile.
3. Doppio clic ed inserire il codice.

```
Sub LoadItem_Click ()
```

È importante sapere se l'utente ha fatto clic sul pulsante Annulla; per questo motivo, si utilizzerà l'errore generato. Si utilizza il controllo degli errori nel modo seguente.

```
On Error Resume Next
CMDialog1.CancelError = -1
```

Inoltre, si può specificare il titolo della finestra di dialogo nel modo seguente.

```
CMDialog1.DialogTitle = "Apri file"
```

Si può specificare l'estensione per i file visualizzati inizialmente nella finestra di dialogo.

```
CMDialog1.Filter = "Solo testo | *.txt"
```

Inoltre è possibile specificare le impostazioni per l'apertura ed il salvataggio dei file.

OFN_READONLY	Seleziona la casella Sola lettura.
OFN_OVERWRITEPROMPT	Messaggio se il file esiste, l'utente può sovrascrivere.
OFN_HIDEREADONLY	Nasconde la casella Sola lettura.
OFN_NOCHANGEDIR	Disabilita il cambiamento di directory.
OFN_SHOWHELP	Inserisce il pulsante della guida.
OFN_ALLOWMULTISELECT	Consente la selezione di file multipli.
OFN_PATHMUSTEXIST	L'utente può specificare solo un percorso valido.
OFN_CREATEPROMPT	Richiede se l'utente vuole creare il file se non esiste.
OFN_SHAREWARE	Ignora l'errore OFN_SHARINGVIOLATION.
OFN_NOREADONLYRETURN	Il file non è a sola lettura.

```
CMDialog1.Flags = OFN_READONLY
```

Ora si visualizza la finestra di dialogo.

```
CMDialog1.Action = 1
```

Infine, dopo che l'utente ha chiuso la finestra di dialogo, si verificherà che non abbia fatto clic su Annulla e si legge il file.

```
If Err = 0 Then
        Open CMDialog1.Filename For Input As #1
'LOF (Length Of File) numero di byte presenti in un file
        PadText.Text = Input$(LOF(1), #1)
        Close #1
```

```
        End If
    End Sub
```
Nello stesso modo si può creare una finestra di dialogo per il salvataggio dei file.
```
    Sub SaveItem_Click ()
        On Error Resume Next
        CMDialog1.CancelError = -1
        CMDialog1.DialogTitle = "Salva file"
        CMDialog1.Filter = "Solo testo | *.txt"
        CMDialog1.Flags = OFN_READONLY
        CMDialog1.Action = 2
        If Err = 0 Then
            Open CMDialog1.Filename For Output As #1
            Print #1, PadText.Text
            Close #1
        End If
    End Sub
```
Nello stesso modo si può creare una finestra di dialogo per la stampa.
```
    Sub PrintItem_Click ()
        On Error Resume Next
```
Si verifica se è stato premuto Annulla; controlla l'errore.
```
        CMDialog1.CancelError = -1
        CMDialog1.FromPage = 1          'pagina da cui inizia la stampa
        CMDialog1.ToPage = 1            'pagina di fine stampa: una sola!
        CMDialog1.Action = 5
        If Error = 0 Then
            Printer.Print PadText.Text
            Printer.EndDoc
        End If
    End Sub
```
I flags che si possono utilizzare sono i seguenti.

PD_ALLPAGES	Imposta il pulsante Tutto.
PD_SELECTION	Imposta il pulsante Selezione.
PD_PAGENUMS	Imposta il pulsante Pagine.
PD_NOSELECTION	Non consente l'uso del pulsante Selezione.
PD_NOPAGENUMS	Non consente l'uso del pulsante Pagine.
PD_COLLATE	Imposta il pulsante Fascicola copie.
PD_PRINTTOFILE	Imposta la casella Stampa su file.
PD_PRINTSETUP	Visualizza la finestra Imposta stampante
PD_NOWARNING	Disabilita i messaggi di avviso.
PD_RETURNDC	Restituisce un handle (maniglia) hDC.
PD_RETURNIC	Restituisce un information context.
PD_SHOWHELP	Visualizza il pulsante guida.
PD_USEDEVMODECOPIES	Non abilita la casella Numero di copie.
PD_HIDEPRINTTOFILE	Nasconde la casella Stampa su file.

Nello stesso modo si può creare una finestra di dialogo per la selezione del colore del testo.
```
    Sub ColoreItem_Click ()
        CMDialog1.Color = PadText.ForeColor
        CMDialog1.Action = 3
        PadText.ForeColor = CMDialog1.Color
    End Sub
```
I flags che si possono utilizzare sono i seguenti.

CC_RGBINIT	Valore iniziale del colore.
CC_FULLOPEN	Include i colori personalizzati.

CC_PREVENTFULLOPEN	Non consente di aprire interamente la finetsra.
CC_SHOWHELP	Visualizza il pulsante della guida.

Nello stesso modo si può creare una finestra di dialogo per la selezione dei font.

```
Sub FontItem_Click ()
    On Error Resume Next
    CMDialog1.CancelError = -1                    'True
```

Bisogna prima caricare tutte le proprietà correnti del font in CMDialog in modo da potere visualizzare i default.

```
    CMDialog1.FontName = PadText.FontName
    CMDialog1.FontItalic = PadText.FontItalic
    CMDialog1.FontStrikethru = PadText.FontStrikethru
    CMDialog1.FontSize = PadText.FontSize
    CMDialog1.FontUnderline = PadText.FontUnderline
    CMDialog1.Color = PadText.ForeColor
```

Ora si può impostare la proprietò Flags.

```
    CMDialog1.Flags = CF_SCREENFONTS Or CF_EFFECTS
    CMDialog1.Action = 4
```

A questo punto, la finestra è visualizzata, ma se l'utente preme Annulla non si dovranno trasferire le impostazioni; basta mettere la proprietà CancelError = True

```
    If Err = 0 Then
        PadText.FontName = CMDialog1.FontName
        PadText.FontItalic = CMDialog1.FontItalic
        PadText.FontStrikethru = CMDialog1.FontStrikethru
        PadText.FontSize = CMDialog1.FontSize
        PadText.FontUnderline = CMDialog1.FontUnderline
        PadText.ForeColor = CMDialog1.Color
    End If
End Sub
```

Le impostazioni possibili per la proprietà Flags sono le seguenti.

CF_SCREENFONTS	Selezione dei soli font per il video.
CF_PRINTERFONTS	Selezione dei soli font per la stampante.
CF_BOTH	Selezione di entrambi i tipi di font.
CF_SHOWHELP	Visualizza il pulsante della guida nella finestra.
CF_EFFECTS	Selezione degli effetti.
CF_APPLY	Abilita il pulsante Apply.
CF_ANSIONLY	Selezione dei soli font che utilizzano il set ANSI.
CF_NOVECTORFONTS	Non consente l'uso dei font vettoriali.
CF_NOSIMULATIONS	Non consente la simulazione dei font GDI.
CF_LIMITSIZE	Limita la dimensione dei font.
CF_FIXEDPITCHONLY	Uso dei soli font a dimensione fissa.
CF_WYSIWYG	Uso dei soli font sia per il video che per la stampante.
CF_FORCEFONTEXIST	Uso dei soli font esistenti.
CF_SCALABLEONLY	Uso dei soli font scalabili.
CF_TTONLY	Uso dei soli font TrueType.

GUIDA (HELP/?)

1. Creazione della Guida: *.RTF

La creazione del file degli argomenti comporta la stesura del testo che sarà visualizzato quando l'utente attiva la Guida e l'inserimento di codici di controllo che consentono i collegamenti ipertestuali, è un file creato in Word e salvato come RTF (Rich Text Format). All'interno della Guida, è possibile consultare argomenti in due modi: selezionando uno degli argomenti sottolineati si passa ad un'altra finestra; oppure selezionando un argomento sottolineato da una linea tratteggiata si visualizza una finestra a comparsa.

Per sottolineare una parola:

1. evidenziarla;
2. Formato/Carattere/Sottolineatura Doppia;
3. posizionare il cursore subito dopo la parola (non ci devono essere spazi);
4. Formato/Carattere/Nascosto
5. inserire il contrassegno, indica a quale argomento bisogna saltare (se è seguito da @pathname salta ad una altro file, invece >windowname salta ad un'altra finestra);
6. Inserisci/Interruzione Di pagina;
7. bisogna indicare che si tratta della destinazione indicata al punto 5, per fare questo posizionare il cursore sulla nuova pagina e si selezioni Inserisci/Note personalizzata, si digiti **#** (identifica in modo univoco un argomento) ed il contrassegno. A questo punto Word divide la finestra in due parti, visualizzando nella parte inferiore le note e nella parte superiore il cursore dopo # che appare nel testo che si sta impostando;
8. File/Salva con nome/*.RTF

Per sottolineare una parola con una linea tratteggiata:

1. evidenziarla;
2. Formato/Carattere/Sottolineatura Singola;
3. posizionare il cursore subito dopo la parola (non ci devono essere spazi);
4. Formato/Carattere/Nascosto
5. inserire il contrassegno, indica a quale argomento bisogna saltare;
6. Inserisci/Interruzione Di pagina;
7. posizionare il cursore sulla nuova pagina e si selezioni Inserisci/Note personalizzata, si digiti # con il commento della finestra a comparsa;
8. File/Salva con nome/*.RTF

Per ciascun argomento si deve assegnare oltre al contrassegno, anche un titolo ed un insieme di parole chiave, il simbolo **$** assegna alla pagina corrente un titolo all'argomento che apparirà nella casella di riepilogo quando l'utente utilizza l'opzione Cerca della Guida. Il simbolo **K** definisce una parola chiave che l'utente può utilizzare nella ricerca degli argomenti (più parole chiave sono separate dal punto e virgola). Il simbolo **+** definisce la successione degli argomenti, facoltativo.

L'inserimento d'immagini avviene in due modi:

1. diretto nel file *.RTF;
2. in fase di compilazione mediante {bmc pathname}.

In Word non usare l'apostrofo, ma Inserisci/Simbolo/Testo normale.

Nel titolo inserire $, in questo modo nella cronologia sparisce: argomento senza titolo.

2. Creazione del file di progetto: *.HPJ

È in formato ASCII, con struttura simile ai file *.INI, in pratica una serie di sezioni che determinano l'aspetto del file di Guida ed alcuni parametri necessari alla compilazione.

```
[OPTIONS]
errorlog = filename.err     ;salva gli errori in fase di compilazione
title = ... GUIDA           ;titolo della Guida
compress = false            ;con true il file è compattato
warning = 3                 ;visualizza tutti i messaggi di avvertimento
contents = contents ;nome della prima finestra che contiene il sommario argomenti
[FILES]
filename.rtf                ;il file che contiene il testo della Guida
[WINDOWS]
main = "titolo",(0,0,1023,1023),,,(192,192,192)
;si possono specificare titolo, posizione e dimensione della finestra
[CONFIG]
[MAP]
```

3. Creazione del file di Guida: *.HLP

Si utilizza HC31.EXE passandogli in input il file HPJ, in output si avrà il file HLP.

4. Collegamento del file HLP all'applicazione

Si aggiunge il menu Guida, con le seguenti voci:

CAPTION = &? NAME = Guida
    CAPTION = &Sommario NAME = SommarioItem
    CAPTION = &Cerca argomenti... NAME = CercaItem
    CAPTION = &Uso della Guida NAME = UsoItem
    CAPTION = - NAME = Separatore
    CAPTION = Informazioni su... NAME = InformazioniItem

L'allineamento a destra della voce ? è realizzato durante il run-time in:

```
Sub Form_Load ()
    GuidaI.Caption = Chr(8) & Guida.Caption
End Sub
```

Inserire il codice in SommarioItem:

```
Sub SommarioItem_Click ()
    Const HELP_INDEX = &H3
    CMDialog1.HelpFile = "c:\vb\samples\max\file_seq.hlp"
    CMDialog1.HelpCommand = HELP_INDEX
    CMDialog1.HelpKey = "MouseDown"
    CMDialog1.Action = 6
End Sub
```

Inserire il codice in CercaItem:

```
Sub CercaItem_Click ()
    Dim ret As Integer, lpNullStr As Long
    ret = Winhelp(Me.hWnd, "c:\vb\esempi\file_seq\file_seq.hlp", 261, lpNullStr)
End Sub
```

Inserire il codice in UsoItem:

```
Sub UsoItem_Click ()
    Const HELP_HELPONHELP = &H4
    CMDialog1.HelpFile = "c:\windows\winhelp.hlp"
    CMDialog1.HelpCommand = HELP_HELPONHELP
    CMDialog1.HelpKey = "MouseDown"
    CMDialog1.Action = 6
End Sub
```

Inserire il codice in InformazioniItem;

```
Sub InformazioniItem_Click ()
    RetVal% = MsgBox("FILE SEQUENZIALI IN VISUAL BASIC!", 64, "About")
End Sub
```

Un altro metodo è quello di utilizzare la funzione:

```
Declare Function WinHelp Lib "User" (ByVal hWnd As Integer, ByVal lpHelpFile As String,
                                     ByVal wCommand As Integer, dwData As Any) As Integer
```

dove gli argomenti hanno il seguente significato:

hWnd l'handle della finestra principale
lpHelpFile il pathname del file della Guida
wCommand sono le seguenti costanti

```
Global Const HELP_CONTEXT = &H1  visualizza l'argomento identificato da dwData
Global Const HELP_QUIT = &H2     chiude la Guida
Global Const HELP_INDEX = &H3    visualizza l'indice
Global Const HELP_HELPONHELP = &H4  visualizza la Guida sull'uso della Guida
Global Const HELP_SETINDEX = &H5    imposta un sommario differente
Global Const HELP_KEY = &H101 visualizza l'argomento  per la parola chiave in dwData
Global Const HELP_MULTIKEY = &H201 cerca la parola chiave in una tabella differente
```

dwData un long integer impostato a zero

Per aggiungere la dichiarazione della funzione WinHelp() è sufficiente caricare, nel modulo

*.BAS, il file: File/Load Text c:\vb\hc\winhelp.txt

FILE

Esistono cinque modi di aprire i file:

1. Input sequenziale
2. Output sequenziale
3. Append sequenziale
4. Input/Output casuale
5. Input/Output binario

Le prime tre modalità si riferiscono ai file sequenziali, normalmente utilizzati per il testo; la quarta ai file random, sono pronti sia per l'input che per l'output ma i dati devono essere suddivisi in record; la quinta ai file binari, non ci si preoccupa d'interpretare il contenuto del file, per esempio gli eseguibili (EXE) trattati sempre byte per byte.

ACCESSO	ISTRUZIONI
Sequenziale	*Open, Input$, Line Input #, Print #, Write #, Close*
Casuale	*Type...End Type, Open, Get #, Put #, Len, Close*
Binario	*Open, Input$, Get #, Put #, Seek, Close*

Apertura di file

L'enunciato Open segue la seguente sintassi:

Open fff$ [For mmm] [Access aaa] [lll] As [#] nnn% [Len = rrr%]

dove gli argomenti hanno il seguente significato:

- fff$ il nome di file, incluso l'eventuale percorso
- mmm modalità: Input, Output, Append, Random, Binary
- aaa accesso: Read, Write, Read Write
- lll blocco (limita l'accesso al file da parte di altre applicazioni): Shared, Lock, Read, Lock Write, Lock Read Write
- nnn% numero di riferimento del file (1..255)
- rrr% dimensione del record per i file random, o del buffer per i sequenziali.

Se il file non esiste e si tenta di aprirlo, Visual BASIC lo crea automaticamente. D'altro canto, se si apre un file già esistente per l'output e vi si scrive, il contenuto originale del file è distrutto a meno che si usi l'append.

Chiusura di file

Close # nnn%

Se si usa Close senza specificare un numero di file, chiude tutti i file aperti.

Lettura di file sequenziali

I metodi standard per leggere sono:

Input # nnn%, listaespressioni
Line Input # nnn%, variabilestringa
Input$ (bbb%, [#] nnn%)

dove gli argomenti hanno il seguente significato:

- listaespressioni è un elenco di esprressioni (stringhe) che si desidera scrivere
- bbb% è il numero di byte da leggere
- variabilestringa è il nome della variabile dove sono inseriti i dati.

Input deve trovare gli elementi del file separati da virgole, spazi o caratteri di fine riga (ritorno a capo): non adatta a leggere un testo.

Line Input legge le stringhe dal file fino a quando incontra un carattere di fine riga, ciò significa che si dovrebbe leggere ogni riga del file (nel caso sia diviso in righe) separatamente nel modo seguente.

Do Until EOF (1)

Line Input #1,Stringa$
Form1.PadText.Text=Form1.PadText.Text+Stringa$+Chr$(13)+Chr$(10)
Loop

Bisogna aggiungere CR e LF perché Line Input considera questi due caratteri semplicemente come separatori tra le stringhe e li cancella. Dal momento che fanno parte del file, li si reinserisce.
Input$ è specificamente dedicato alla lettura delle stringhe perché non sopprime i caratteri di ritorno a capo e di fine riga; tuttavia, si deve indicare esattamente il numero di byte che si desidera leggere. Il numero di byte da leggere è semplicemente la lunghezza del file espressa in byte, e si usa la funzione, LOF(), che restituisce proprio questo dato.

Form1.PadText.Text = Input$ (LOF (1),#1)

Input$ può leggere dei file lunghi al massimo 32767 byte se si apre il file per un accesso di tipo sequenziale o binario. Tuttavia, se si devono utilizzare dei file più lunghi, si può rilevare la loro dimensione tramite LOF(), ed effettuare quindi delle letture ripetute in successione fino a leggere tutti i dati necessari.

Scrittura di file sequenziali

I metodi standard per scrivere sono:

Write # nnn%, listaespressioni
Print # nnn%, listaespressioni

Write inserisce delle virgole per separare le espressioni, racchiude le stringhe tra virgolette quando li scrive nel file ed inserisce una nuova riga (vuota) alla fine del file.
Print invia una sola lunga stringa al file.

FILE RANDOM

Per memorizzare grandi quantità d'informazioni o se le informazioni sono strutturate in record si usano i file random che consentono tempi di accesso e di ricerca minori.
Un file sarà quindi costituito da un certo numero di record, tutti della stessa lunghezza.
Aprire un nuovo progetto con la form CAPTION = Database, inserire quindi tre Textbox:
CAPTION = Cognome NAME = CognomeField
CAPTION = Nome NAME = NomeField
CAPTION = Note NAME = NoteField MULTILINE = TRUE SROLLBARS = VERTICAL
e tre Label con gli stessi identificatori. Inserire un menu:
CAPTION = &File NAME = FileMenu

CAPTION = &Inserisci record... NAME = AddAnItem
CAPTION = &Trova record... NAME = FindItem
CAPTION = &Apri... NAME = LoadItem
CAPTION = Sa&lva con nome... NAME = SaveItem
CAPTION = - NAME = Separator
CAPTION = &Esci NAME = ExitItem

Il menu di Guida è uguale all'applicazione file sequenziali.
File/New Module/file_ran.bas

Type Record
*Cognome As String * 30*
*Nome As String * 20*
*Note As String * 200*
End Type
Global TheData(100) As Record
Global TotalRecords As Integer
Global NumberFileRecords As Integer

Inserire il codice in File/Inserisci record:

Sub AddAnItem_Click ()
TotalRecords = TotalRecords + 1

TheData(TotalRecords).Cognome = CognomeField.Text
TheData(TotalRecords).Nome = NomeField.Text
TheData(TotalRecords).Note = NoteField.Text

Ora che i dati sono stati memorizzati si deve inserire il cognome nella casella di riepilogo con NAME = Cognome List (che è ordinata alfabeticamente in modo automatico) in modo che l'utente possa selezionare i record agevolmente. A tal fine, si usa il metodo:

Form2.CognomeList.AddItem CognomeField.Text
NomeField.Text = "":CognomeField.Text = ""NoteField.Text = ""
CognomeField.SetFocus
End Sub

In generale, il metodo AddItem segue questa sintassi:

Form.casella di riepilogo.AddItemstringa$[,indice]

dove indice specifica la posizione del nuovo inserimento nella casella di riepilogo, se è ordinata alfabeticamente non è necessario specificare l'indice per l'inserimento.
Per rimuovere una voce, si usa il metodo:

Form.casella di riepilogo.RemoveItem,indice

In questo caso l'indice lo si deve usare per specificare la voce che si desidera rimuovere.
Inserire il codice in File/Trova record...

Sub FindItem_Click ()
Form2.Show
End Sub

Si proceda ora alla creazione della finestra di dialogo con File/New Form:
CAPTION = Trova record... MINBUTTON = FALSE MAXBUTTON = FALSE
BORDERSTYLE = FIXED SINGLE
su cui si crea la casella di riepilogo con Listbox NAME = CognomeList SORTED = TRUE.

Le **caselle di riepilogo** sono usate quando si ha un ampio ventaglio di scelte da proporre e si vuole limitare l'utente a queste scelte, una proprietà importante è SORTED, ordina tutte le voci presenti.

Le **caselle combinate** (Combobox) riuniscono in sé le proprietà delle caselle di riepilogo e delle caselle di testo, consentendo così all'utente di digitare qualsiasi testo o di selezionare una delle scelte disponibili.
Possono essere di tre stili.

1. Caselle combinate standard (0), in cui la freccia è staccata dalla casella di testo;
2. uguale alle prime (1) tranne per il fatto che l'elenco rimane sempre visualizzato;
3. la freccia è attaccata alla casella (2), è in effetti una casella di riepilogo a tendina per cui l'utente non può modificare le scelte.

Si aggiungano due Commandbutton OK e Annulla.
In Annulla inserire il seguente codice:

Sub CancelButton_Click ()
Form2.Hide
End Sub

In generale, si può determinare la voce selezionata in una casella di riepilogo usando le seguenti proprietà:

- ✓ Text, la voce correntemente selezionata;
- ✓ List, matrice di stringhe contenente tutte le voci;
- ✓ ListIndex, l'indice della voce selezionata;
- ✓ ListCount, il numero totale delle voci dell'elenco.

Da ricordare che Text = List(ListIndex).
In OK inserire il seguente codice:

Sub OKButton_Click ()
Call GetItem
End Sub

Gli eventi per le caselle di riepilogo sono due.
1. Click() che si verifica quando l'utente effettua una selezione;
2. DblClick() che si verifica quando l'utente effettua una scelta.

Ciò significa che fare doppio clic su una voce equivale a premere OK dopo averla selezionata, si può quindi aggiungere la stessa procedura a CognomeList_DblClick():

```
Sub CognomeList_DblClick ()
    Call GetItem
End Sub
```

La procedura GetItem è inserita in file_ran.bas, si deve però notare che ogni volta che si usa un riferimento ad un controllo di una form, si deve includere il nome, dato che questo codice non è collegato ad alcuna form.

```
Sub GetItem ()
For Loop_Index% = 1 To 100
'Rtrim$ elimina gli spazi sulla destra di una stringa, Ltrim$ sulla sinistra.
If (RTrim$(TheData(Loop_Index%).Cognome) = RTrim$(Form2.CognomeList.Text))
Then Exit For
Next Loop_Index%
Form1.CognomeField.Text = TheData(Loop_Index%).Cognome
Form1.NomeField.Text = TheData(Loop_Index%).Nome
Form1.NoteField.Text = TheData(Loop_Index%).Note
Form2.Hide
End Sub
```

Si effettua un confronto tra il cognome selezionato e tutti i cognomi del file.
Quando si trova quello desiderato (che deve essere nel file, dato che la casella di riepilogo si limita a visualizzare quelli inseriti), si esce dal ciclo.
A questo punto, si possono riempire i campi. Inserire il codice in File/Salva con nome...

```
Sub SaveItem_Click ()
    On Error Resume Next
    CMDialog1.CancelError = -1
    CMDialog1.DialogTitle = "Salva file"
    CMDialog1.Filter = "Solo dati|*.dat"
    CMDialog1.Flags = OFN_READONLY
    CMDialog1.Action = 2
    If Err = 0 Then
        Open CMDialog1.Filename For Random As #1 Len = Len(TheData(1))
        For Loop_Index% = 1 To TotalRecords
            Put #1, , TheData(Loop_Index%)
        Next Loop_Index%
        Close #1
    End If
End Sub
```

Inserire il codice in Apri...

```
Sub LoadItem_Click ()
    On Error Resume Next
    CMDialog1.CancelError = -1
    CMDialog1.DialogTitle = "Apri file"
    CMDialog1.Filter = "Solo dati|*.dat"
    CMDialog1.Flags = OFN_READONLY
    CMDialog1.Action = 1
    If Err = 0 Then
        Open CMDialog1.Filename For Random As #1 Len = Len(TheData(1))
```

si deve per prima cosa rilevare il numero di record presenti nel file, il che è possibile dividendo la lunghezza del file in byte per la dimensione del record

```
        NumberFileRecords = LOF(1) / Len(TheData(1))
```

```
            For Loop_Index% = 1 To NumberFileRecords
                Get #1, , TheData(Loop_Index%)
            Next Loop_Index%
            Close #1
        End If
```
adesso si devono caricare i cognomi dei record nella finestra Trova record... però prima si devono cancellare tutti gli inserimenti correnti
```
        For Loop_Index% = 1 To TotalRecords
            Form2.CognomeList.RemoveItem 0
        Next Loop_Index%
        TotalRecords = NumberFileRecords
        For Loop_Index% = 1 To TotalRecords
            Form2.CognomeList.AddItem TheData(Loop_Index%).Cognome
        Next Loop_Index%
        Form2.Show
End Sub
```
La funzione Len() restituisce la lunghezza in byte della dimensione del record.

Scrittura di file ad accesso casuale

Il metodo standard per scrivere è:

Put [#] nnn%, [rrr%], vvv%

dove gli argomenti hanno il seguente significato:

- nnn% è il numero del file (1..255)
- rrr% è il numero del record che si vuole inserire nel file, se non si specifica è inserito in append
- vvv% è la variabile record da scrivere.

Lettura di file ad accesso casuale

Il metodo standard per leggere è:

Get [#] nnn%, [rrr%], vvv%

dove gli argomenti hanno il seguente significato:

- rrr% è il numero del record che si vuole leggere nel file, se non si specifica è letto il record successivo a quello corrente
- vvv% è la variabile record da leggere.

Posizionamento all'interno di un file ad accesso casuale

Il metodo standard per posizionarsi all'interno di un file è:

Seek [#] nnn%, ppp&

l'istruzione permette di specificare il record da leggere o scrivere, dove ppp& (long integer) è la posizione nel file del record. Per i file sequenziali, ppp& è espresso in byte; per i file random, in numeri di record. In pratica:

Get #1,Loop_Index%, TheData(Loop_Index%)

equivale a:

```
Seek #1, Loop_Index%
Get #1, , TheData(Loop_Index%)
```

Funzione FreeFile

Se, in un certo momento della nostra applicazione, non si conosce il numero dei file aperti, possiamo utilizzare questa funzione: restituisce un intero che costituisce il primo identificatore libero che possiamo utilizzare.

```
Dim N As Integer
N = FreeFile
```

Cancellazione logica e fisica di un record

La rimozione di record da un file non è permessa, allora per effettuare la cancellazione logica di record bisogna sovrascriverli con un record che non contiene alcun valore, ma lo spazio occupato non è liberato.

Per rimuovere fisicamente i record cancellati è necessario:

1. ricopiare in un nuovo file tutti i record non cancellati
2. eliminare il vecchio file: *Kill nomefile*
3. rinominare il nuovo con il nome del file preesistente: *Name vecchiofile As nuovofile*

UBERTINI MASSIMO
http://www.ubertini.it
massimo@ubertini.it

Dip. Informatica Industriale
I.T.I.S. "Giacomo Fauser"
Via Ricci, 14
28100 Novara Italy
tel. +39 0321482411
fax +39 0321482444
http://www.fauser.edu
massimo@fauser.edu