Ancora Visual Basic for Application

di Francesco Petroni e Raffaele Valensise

Nello scorso numero di MC abbiamo presentato un primo articolo sul Visual Basic Application Edition, presente come ulteriore linguaggio di programmazione nel nuovo Excel 5.0.
Lo scopo dell'articolo era quello di introdurre questo nuovo strumento di programmazione, che, secondo i piani della Microsoft, dovrebbe costituire a breve l'unico linguaggio comune a tutti i suoi applicativi per Windows, non solo perché avrà una base uguale per tutti, ma anche perché permetterà, tramite la tecnologia OLE Automation, di sviluppare procedure, scritte in VBA, che lavorino spaziando su più applicativi

Si tratta di un progetto a dir poco ambizioso, che ha come obiettivo finale quello di rendere l'ambiente Windows, l'intero ambiente comprendente tutto quello che è gestito da Windows o che è gestibile da Windows, dalle funzionalità generali dell'ambiente fino alle singole funzionalità di ciascun applicativo, un unico sistema integrato.

Prima di inizare a vedere gli esercizi proposti (il taglio dell'articolo è, al solito, assolutamente pratico) facciamo un breve riassunto, per punti, della puntata precedente.

Prima di tutto va ribadito il fatto che VBA affianca, nella versione 5.0 di Excel, il vecchio linguaggio Macro, per poi sostituirlo completamente, con le prossime versioni.

Il VBA permette, al pari del vecchio linguaggio, di usare il registratore di macro, che oltre a rendere molto facile la memorizzazione di procedure ripetitive, è anche un efficace strumento per lo studio delle istruzioni, molte delle quali immediatamente sperimentabili proprio con il registratore.

Due aspetti negativi con i quali si scontra l'utente pratico delle macro delle precedenti versioni di Excel, consistono nel fatto che non esiste un convertitore di codici (anche a causa della filosofia assolutamente differente tra i due sistemi), per cui mentre in Excel 5.0 si possono continuare ad usare le vecchie macro Excel 4.0, non è possibile convertirle direttamente in macro VBA (operazione che sarà indispensabile quando si passerà alla 6.0) e nel fatto che Microsoft ha «approfittato» del cambiamento per eliminare il manuale di riferimento per le funzioni, molto voluminoso ed efficace per chi sviluppava alla vecchia maniera, sostituito da un più smilzo Manuale dell'Utente di Visual Basic, che tratta in maniera sufficientemente approfondita gli argomenti generali di programmazione, ma che non contiene l'elenco alfabetico dei comandi.

Le istruzioni, i comandi equivalenti, le proprietà, i metodi, gli oggetti, le parole chiave, insomma gli elementi che vanno conosciuti «per nome» dal programmatore VBA, sono 2.177, duemilacentosettantasette, come si può verificare dal file in dotazione VBARIF.XLS, che contiene l'elenco completo di tali parole nella versione inglese e in quella italiana.

Evidentemente tanto più è grande il numero di «voci» tanto più costa tradurli in un manuale, e evidentemente tanto più servirebbe un manuale.

L'organizzazione della Cartella di Lavoro

Prima di cominciare ripetiamo, per permettere di seguire l'articolo anche a chi non avesse letto il precedente, come è organizzata la nuova cartella di lavoro di Excel 5.0 che ingloba le vecchie tipologie di foglio e di file.

Con Excel 5.0 si lavora su Cartelle di Lavoro.

Il file con la cartella di lavoro ha desinenza XLS (come i vecchi file Excel 2.0, 3.0, 4.0).

Una cartella di lavoro però contiene fogli, da 1 fino ad un massimo di 256. I fogli vengono fisicamente evidenziati dalla classica linguetta, che mostra il nome del foglio.

Esistono numerosi comandi per gestire i fogli (nominare, evidenziare, inserire, cancellare, spostare, ecc.). È anche possibile scrivere formule a cavallo di più fogli (es. la somma della cella A1 di dodici fogli).

Esistono cinque tipi di foglio:
– Foglio Dati tradizionale. È lo spreadsheet classico;
– Foglio Macro alla vecchia maniera. Assomiglia al precedente solo che le colonne appaiono più larghe e che ci si possono inserire le funzioni di programmazione, che sono differenti da quelle inseribili in un foglio normale;
– Foglio Dialog Box. È nuovo. È un foglio a quadretti piccoli, in cui si può inserire il disegno di una dialog box (una sola per foglio). Il foglio dialog box in pratica sostituisce il vecchio Dialog Box Editor. Da una parte permette, come vedremo tra un po', di disegnare delle box più ricche (ad esempio c'è anche l'elemento Scroll Bar), e dall'altra facilita l'uso, da parte del programma, della box che viene identificata semplicemente dal suo nome;
– Foglio Grafico. Un grafico prodotto con le nuove potentissime funzionalità di Excel 5.0 può essere piazzato su un foglio Dati o su un foglio di tipo Grafico;
– Foglio Modulo. È il foglio nel quale vanno scritti i programmi VBA, questi

possono essere delle funzioni personalizzate oppure delle procedure.

Al lancio di Excel viene proposta una Cartella con 16 fogli dati. Essendo tale valore impostabile come opzione generale conviene subito portarlo ad 1, tanto poi è comunque possibile inserire i fogli in più, quando effettivamente servano.

Cosa faremo nell'articolo

L'articolo ha una finalità didattica ed ha un taglio pratico. Si può dividere in quattro parti. Questa, che anticipa il significato dei quattro esercizi proposti. La seconda che introduce alcuni argomenti generali legati al nuovo modo di programmare. Segue la descrizione, più dettagliata, degli esercizi. Infine le necessarie conclusioni.

Tornando agli esercizi, il primo ci permette di sperimentare i comandi di selezione, importantissimi in un linguaggio che deve comunque «fare i conti» con le celle del foglio.

Il secondo introduce il concetto, importantissimo, di Finestra di Dialogo personalizzata, strumento principale di interfaccia tra l'utente e i dati sul foglio. Excel 5.0 dispone di uno speciale tipo di foglio (tipo Finestra di Dialogo) che sostituisce, ampliandone le possibilità, il vecchio Dialog Box Editor.

Nel terzo esercizio realizziamo una minifunzionalità File Apri. Con l'occasione citeremo alcune differenze tra VBA e Visual Basic 3.0.

L'ultimo esercizio è invece un po' più intricato perché consiste nella costruzione di una Dialog Box più complessa, sia perché contiene numerose tipologie di oggetti (che in VBA si chiamano Controlli) sia perché svolge numerose funzioni.

La filosofia degli oggetti

Visual Basic for Application consente di controllare e manipolare qualsiasi elemento di Excel attraverso la definizione di «oggetti» gestibili direttamente da procedura.

Nella concezione Visual Basic è considerato come «oggetto» un qualsiasi elemento che è possibile utilizzare da programma per eseguire delle operazioni.

Ed esempio, «Intervallo» è l'oggetto che serve a identificare un range di celle del foglio di lavoro. Da notare anche che la singola cella non viene considerata oggetto in Excel: per gestire una o più celle occorre sempre utilizzare l'oggetto «Intervallo».

Figura 1 - Excel 5.0 - Visual Basic for Application - Prime procedure in foglio.
Il primo esempio di uso del Visual Basic for Application (VBA per gli amici) consiste in un piccolo gruppo di procedure, richiamabili da alcuni pulsanti posti sul foglio che agiscono su insiemi di celle selezionate. Con l'occasione vediamo come si fa per associare all'oggetto pulsante la Macro. Si seleziona il pulsante e poi facendo click con il tasto destro del mouse si richiama il Quick Menu che prevede, tra le altre, l'opzione Assegna Macro. Il comando di menu invece è Strumenti Assegna Macro.

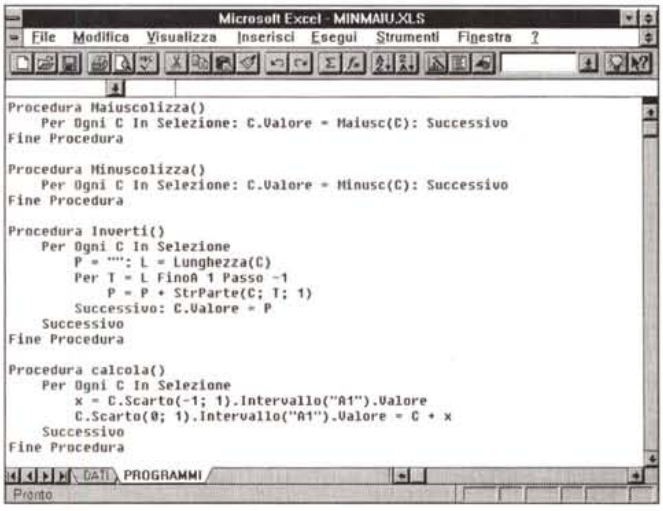

Figura 2 - Excel 5.0 - Visual Basic for Application - Il foglio con i programmi.
Per scrivere i programmi occorre inserire, dopo il foglio normale, un foglio tipo Modulo (comando Inserisci Macro Modulo), in cui si possono scrivere «a manina» le istruzioni di programmazione. La più interessante è la:
Per Ogni C in Selezione
....
Successivo
che permette di eseguire un ciclo che coinvolge tutti gli elementi, in questo caso le celle, selezionati.

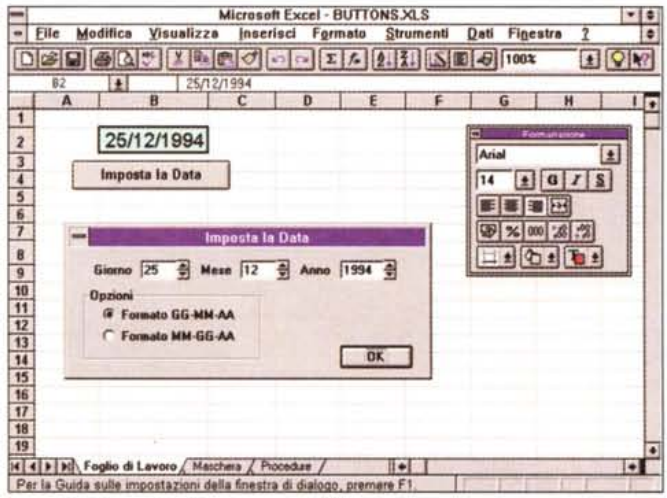

Figura 3 - Excel 5.0 - Visual Basic for Application - Option box e contatori sulla finestra di dialogo.
Passiamo ad un'applicazione che fa uso di una finestra di dialogo e che contiene tre fogli. Quello con i dati dal quale tramite un pulsante si richiama la finestra di dialogo. Gli altri due fogli sono quello che contiene il «disegno» della finestra di dialogo (per attivarlo il comando è Inserisci Macro Finestra di Dialogo) e quello, di tipo Modulo, con le procedure.

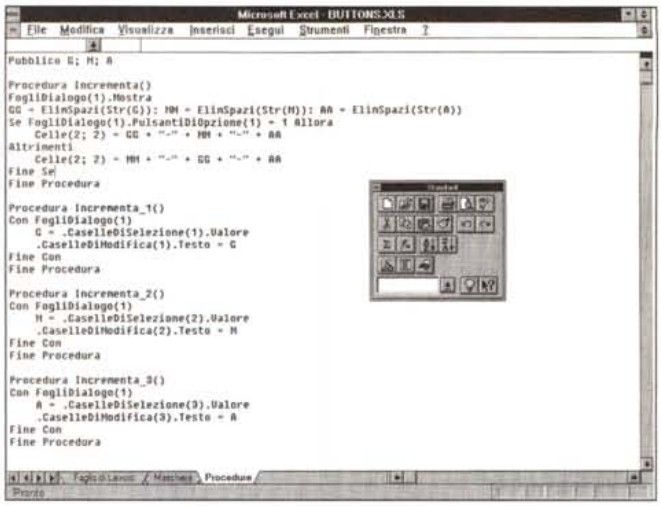

Figura 4 - Excel 5.0 - Visual Basic for Application - Option box e contatori: ecco i programmi.
La nostra finestra di dialogo personalizzata contiene due tipi di oggetti: tre Contatori, realizzati con l'oggetto che VBA chiama Casella di Selezione, che servono per impostare i tre componenti di una data (giorno, mese e anno), e due Pulsanti di Opzione, con i quali si sceglie il formato desiderato per la data impostata.

Ogni oggetto ha delle sue peculiari caratteristiche, chiamate «proprietà», che ne determinano la funzionalità e/o l'aspetto.

L'oggetto Intervallo possiede, fra le altre, sia la proprietà «Altezza», attraverso cui si determina (o si imposta) l'altezza delle celle selezionate, sia la proprietà «Valore», che consente di controllare il contenuto delle celle dell'intervallo. Gli oggetti dispongono inoltre dei cosiddetti «Metodi» che specificano le operazioni che gli oggetti sono in grado di eseguire o di subire. Ad esempio l'Intervallo possiede il metodo «Cancella» che consente di cancellare il contenuto di tutte le celle del range, così come il metodo «Celle» che identifica gli estremi stessi dell'Intervallo.

Poiché in Excel un oggetto può in realtà essere un «insieme» costituito da una serie di oggetti correlati, occorre utilizzare un metodo per identificare un singolo oggetto dell'insieme. L'oggetto-insieme CartellaDiLavoro, possiede infatti il metodo FogliDiLavoro con cui è possibile individuare un singolo foglio:

```
FogliDiLavoro("Foglio1")
```

restituisce per l'appunto il foglio di lavoro «Foglio1». Più in generale, si dever tener presente che alcuni degli oggetti di Excel contengono altri oggetti: il contenitore più grande è l'applicazione Excel stessa (oggetto Applicazione), cui segue la Cartella di Lavoro, che come visto contiene Fogli di Lavoro che a loro volta contengono Intervalli di celle, righe e colonne. In altre parole esiste, nell'applicazione Excel, un'organizzazione gerarchica degli oggetti che va percorsa integralmente quando occorra puntare od individuare un singolo oggetto.

Prime operazioni sugli Oggetti

Per impostare il valore della proprietà di un oggetto occorre specificare entrambi con la sintassi:

```
Oggetto.Proprietà = espressione

Ad esempio:

Celle(1;1).Valore = 18
Celle(1;1).Valore = "Gennaio"
```

Nel primo caso, sfruttando il metodo Celle dell'oggetto Intervallo, impostiamo il valore numerico 18 nella cella A1 (la sintassi prevede la notazione Riga;Colonna), mentre nel secondo caso inseriamo la stringa «Gennaio».

È anche possibile rilevare il valore della proprietà di un oggetto assegnandolo ad una variabile o utilizzarlo per impostare la proprietà di un altro oggetto:

```
MiaVariabile = Celle(1;1).Valore
FogliDiLavoro(«Foglio1»).Celle(1;1).Valore= FogliDiLavoro(«Foglio2»).Celle(5;1).Valore
```

Spesso accade di dover eseguire diverse operazioni su un singolo oggetto, ad esempio per impostare nella cella A1 del foglio di lavoro «Foglio1» il carattere Arial, con gli stili grassetto e corsivo, la dimensione di 16 punti. In questo caso la sintassi per esteso è la seguente:

```
Procedura FormattaCella_Uno()
FogliDiLavoro(1).Celle(1;1).Carattere.Nome = "Arial"
FogliDiLavoro(1).Celle(1;1).Carattere.Dimensione = 16
FogliDiLavoro(1).Celle(1;1).Carattere.Grassetto = Vero
FogliDiLavoro(1).Celle(1;1).Carattere.Corsivo = Vero
Fine Procedura
```

Volendo evitare di ripetere tutti i riferimenti per ogni singolo comando (prassi che finisce per appesantire notevolmente la lettura delle nostre procedure) ci sono due alternative: impostare in una variabile il riferimento di cella, oppure utilizzare l'istruzione Con ... Fine Con. Vediamo come si trasformerebbe il listato nei due casi:

```
Procedura FormattaCella_Due()
Imposta MiaCella = FogliDiLavoro(1).Celle(1;1)
MiaCella.Carattere.Nome = «Arial»
MiaCella.Carattere.Dimensione = 16
MiaCella.Carattere.Grassetto = Vero
MiaCella.Carattere.Corsivo = Vero
Fine Procedura
```

dove l'istruzione Imposta attribuisce alla variabile MiaCella il riferimento alla cella vera e propria piuttosto che al suo contenuto.

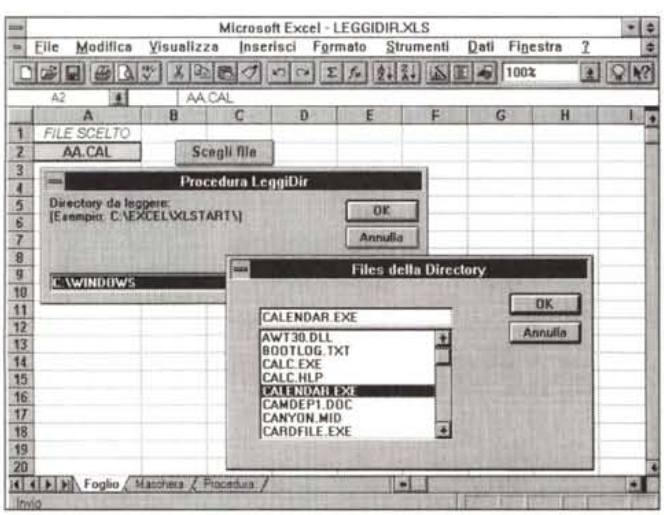

Figura 5 - Excel 5.0 - Visual Basic for Application - Lettura del contenuto di una Directory.
In questo collage vediamo il risultato del nostro programma che lancia una semplice Message Box che chiede il nome di una directory. La funzione Dir(...) contiene il nome di tutti i file letti nella directory. Questi vengono trasferiti nella casella di riepilogo della finestra di dialogo, nella quale è molto semplice selezionare l'elemento desiderato della lista il cui valore viene poi piazzato nella cella A2.

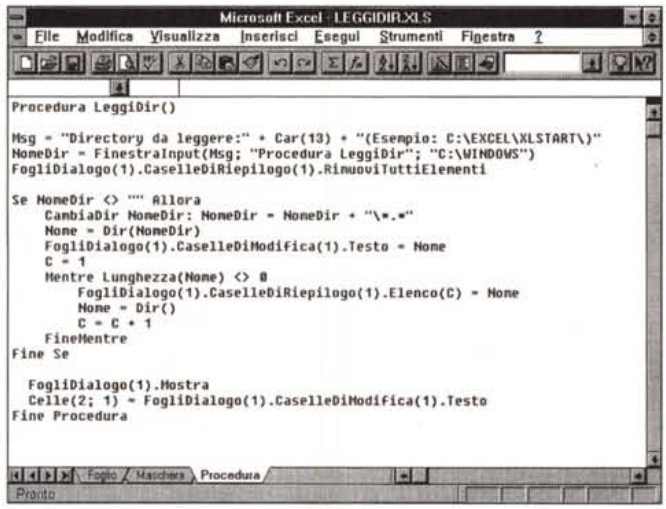

Figura 6 - Excel 5.0 - Visual Basic for Application - I programmi per la lettura del contenuto della directory. Insomma quando occorre richiedere una sola cosa, ad esempio il nome della directory, basta usare una Message Box. Se si richiedono più cose si realizza una Dialog Box, che può contenere più tipi di oggetti. Una grossa novità rispetto alle finestre di dialogo realizzabili con Excel 4.0 e precedenti è costituita dalla possibilità di eseguire dei calcoli, che agiscono su elementi della box stessa, senza dover uscire dalla box. Si può quindi prevedere un pulsante Calcola.

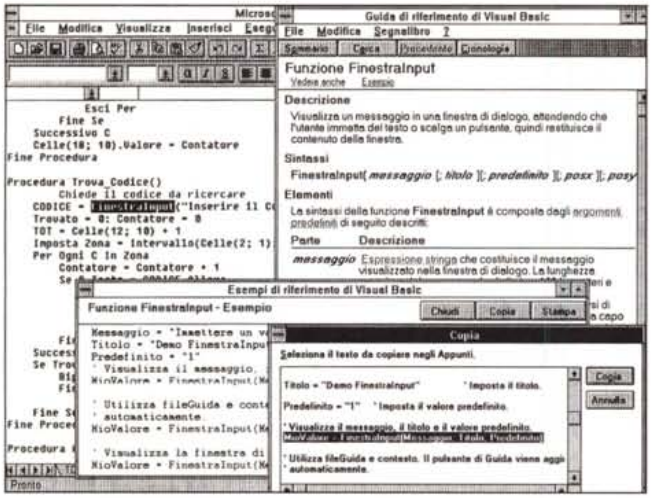

Figura 7 - Excel 5.0 - Visual Basic for Application - Help in linea. Semplicemente evidenziando una parola chiave del VBA e premendo il tasto F1, si attiva il potente help in linea di Excel. In questo caso vediamo le informazioni sulla funzione FinestraInput, comprensive di un utile esempio che è addirittura possibile «copiare» nel nostro listato. L'help è comunque attivabile anche attraverso l'usuale menu «?».

Oppure:

```
Procedura FormattaCella_Tre()
Con FogliDiLavoro(1).Celle(1;1)
.Carattere.Nome = «Arial»
.Carattere.Dimensione = 16
.Carattere.Grassetto = Vero
.Carattere.Corsivo = Vero
Fine Con
Fine Procedura
```

Utilizziamo ora la proprietà CellaAttiva, che restituisce un oggetto Intervallo, per trasformare in maiuscolo il testo contenuto nella cella selezionata:

```
Procedura Maiuscolizza_Cella()
CellaAttiva.Valore = Maiusc(CellaAttiva)
Fine Procedura
```

Operatori matematici, logici e di confronto

In Visual Basic le operazioni che tipicamente si utilizzano per effettuare calcoli fra variabili e costanti hanno una sintassi classica:

```
Importo = 12500
Totale = (Importo * 0,19) + Importo
```

Allo stesso modo è possibile utilizzare i noti operatori logici And, Or e Not (presenti fra le funzioni del foglio di lavoro Excel) cui si aggiungono i forse meno conosciuti Xor, Eqv e Imp, esclusivi del VB:

▼

```
(10>5) And (20>30) restituisce il valore FALSO
(10>5) Or (20>30) restituisce il valore VERO
Not (10>5) restituisce il valore FALSO
(5>10) Xor (20>30) restituisce il valore FALSO, poiché le espressioni si equivalgono
(5>10) Eqv (20>30) restituisce il valore VERO, poiché le espressioni si equivalgono
(5>10) Imp (20<30) restituisce il valore VERO. Darebbe FALSO se la prima espressione
avesse valore VERO e la seconda FALSO.
```

Qualora si volesse disporre anche nel foglio di lavoro Excel di una funzione tipica del Visual Basic (ipotizziamo «Xor») è necessario creare una «Funzione definita dall'utente» così come illustrato nella puntata precedente:

```
Funzione MioXor (espressione1;
espressione2)
MioXor = espressione1 Xor espressione2
Fine Funzione
```

Accanto ai sopra elencati operatori logici fra espressioni, il VB dispone di due operatori di confronto, Fosse e Simile, il primo utilizzato per effettuare confronti fra variabili riferite ad oggetti, il secondo fra stringhe di caratteri:

```
Imposta MiaVar_1 = FogliDiLavoro(1)
Imposta MiaVar_2 = FogliDiLavoro(1)
Risultato = MiaVar_1 Fosse MiaVar_2

restituisce, ovviamente, VERO.

MiaVariabile = «VisualBasic» Simile «V*c»
```

restituisce VERO, poiché la stringa «V*c» (notare il carattere jolly) è contenuta nel testo «VisualBasic». In caso contrario avrebbe restituito il valore FALSO o NULLO (se una delle due stringhe da confrontare fosse vuota).

Costanti e variabili

Rapido accenno alla presenza di alcune costanti incorporate tipiche dell'ambiente Visual Basic e dell'ambiente Excel, le prime caratterizzate dal prefisso «vb», le seconde dal prefisso «xl». Ad esempio, nel contesto della creazione di una finestra-messaggio personalizzata:

```
vbSoloOK significa: Solo pulsante OK;
vbOKAnnulla significa: Pulsanti OK e Annulla.
```

Oppure, per impostare una nuova barra dei menu (come visto nella puntata precedente):

▼

```
xlFoglioDiLavoro significa: Barre dei Menu del Foglio di Lavoro;
xlGrafico significa: Barre dei Menu del Foglio Grafico.
... eccetera, eccetera.
```

Forse più interessante è la possibilità di poter definire costanti «personalizzate», il cui pregio è di essere condivise da tutte le procedure all'interno del modulo (ciò significa che se richiamate da routine di altri moduli, restituiscono il valore 0). L'istruzione Costante serve al caso nostro e va, chiaramente, inserita

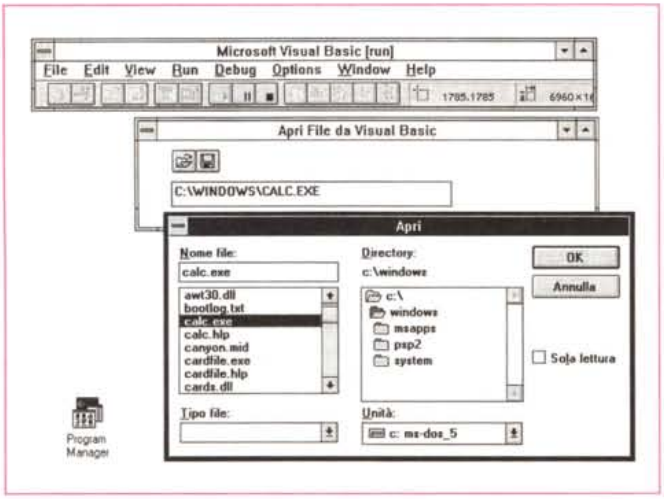

Figura 8 - Excel 5.0 - Visual Basic for Application - Uso delle Common Dialog Box in Visual Basic.
Il linguaggio VBA è a metà strada tra il vecchio linguaggio Macro di Excel e il Visual Basic normale. Ad esempio con il Visual Basic 3.0 è possibile richiamare le finestre di dialogo di Windows, come questa File Apri. Ebbene lo si può fare, lo spieghiamo nel testo, anche da VBA.

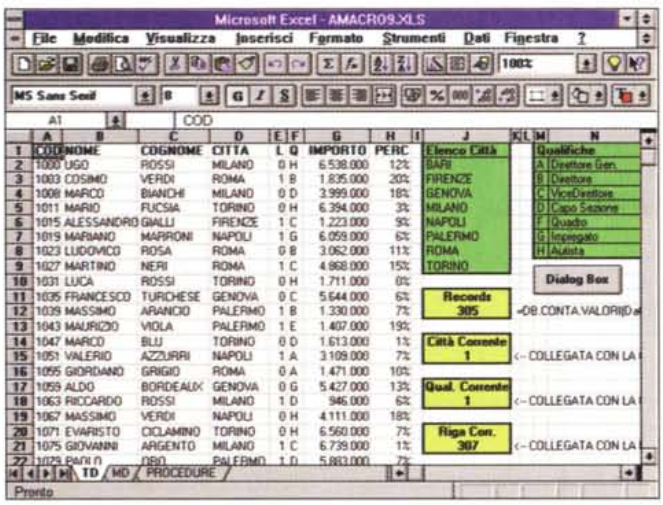

Figura 9 - Excel 5.0 - Visual Basic for Application - Il foglio con i dati.
Il nostro obiettivo è quello di realizzare una finestra di dialogo con la quale interfacciare i dati. Esistono, tra gli altri, due campi gestibili tramite una lista, una lista di città e una di qualifiche, che vengono risolti con due caselle di dialogo. La lista delle qualifiche in realtà consiste in una codifica, codice e descrizione.

prima della definizione di qualsiasi procedura del modulo:

```
Costante Versione_di_MSExcel = 5
Costante Nani_di_Biancaneve = 7
Costante ParolaChiave = "BABALOO"
```

Nel caso delle variabili, è possibile impostare una loro cosiddetta «area di validità», ossia la zona in cui la variabile viene condivisa dal codice Visual Basic, attraverso il tipo di dichiarazione con la quale si definisce la variabile: con Dim, Statica, o Privato all'inizio del modulo la variabile assume validità nel Modulo. Con l'istruzione Pubblico, la variabile ha validità Pubblica. Vediamo di capire meglio il senso dei tre tipi sopra elencati.

Variabile Locale: riconosciuta solo all'interno della routine e si definisce con la dichiarazione Dim o Statica all'interno della routine;

Variabile Modulo: è a disposizione di tutte le routine del modulo e si definisce con la dichiarazione Dim, Statica, o Privato all'inizio del modulo;

Variabile Pubblica: è a disposizione di tutte le routine di tutti i moduli di tutte le cartelle di lavoro e si definisce con la dichiarazione Pubblico prima della definizione delle procedure.

Ecco un esempio che dichiara tutti e tre i tipi di variabile:

```
Pubblico VarGlobale Come Intero Definisce anche
che la variabile sia un intero
Dim
Procedura Calcolo()
Dim VarLocale Come Intero
...
...
Fine Procedura
```

L'istruzione Per ... Ogni

Nella gestione di gruppi di oggetti tipici dell'ambiente MS Excel, l'istruzione Per Ogni ... Successivo rappresenta certamente l'aiuto più valido. Attraverso questo iteratore è infatti possibile impostare dei loop controllati solo dal numero di oggetti compresi nella zona interessata (ad esempio un gruppo di celle, oppure gli oggetti presenti in una finestra di dialogo). Quindi possiamo controllare e impostare le caratteristiche degli elementi tramite una variabile che progressivamente fa riferimento ad ogni oggetto della selezione. Nelle poche righe che seguono utilizziamo la proprietà Selezione per migliorare la procedura Maiuscolizza_Cella(), vista in precedenza, in modo da ottenerne una che metta in maiuscolo i testi contenuti in un gruppo di celle, preventivamente evidenziate con il mouse:

```
Procedura Maiuscolizza_Range()
Per Ogni C In Selezione
C.Valore = Maiusc(C)
Successivo
Fne Procedura
```

Qualche esercizio

Iniziamo con alcune piccole procedure esplorative per poi culminare nella gestione di un elenco tramite maschere di inserimento realizzate ad hoc.

Manipolazione di celle selezionate

Come accennato, la proprietà Selezione riporta in Visual Basic la zona selezionata del foglio di lavoro (sono valide anche le selezioni multiple effettuate in combinazione con il tasto CTRL). Nell'esempio di figura 1 abbiamo scritto 4 piccole procedure che manipolano il contenuto delle singole celle evidenziate abbinandole ad altrettanti pulsanti inseriti nel foglio.

Esaminando il semplice listato in figura 2: le procedure Maiuscolizza() e Minuscolizza() mutano l'eventuale testo contenuto nelle celle in maiuscolo e minuscolo, la procedura Inverti() rovescia testi e numeri, la procedura Calcola() riporta nelle celle adiacenti alla selezione una somma cumulativa del tipo n1; n2+n1; n3+n2+n1; ecc.

L'istruzione Per Ogni ... Successivo in combinazione con il suddetto Selezione si rivela davvero utile. Nella procedura Calcola() infatti si riesce a utilizzare il contatore incrementato dal Per Ogni

per localizzare la cella in cui scrivere il risultato della manipolazione:

```
x = C.Scarto(-1; 1)
C.Scarto(0; 1) = C + x
```

Il metodo Scarto consente di spostarsi inizialmente di 1 riga in alto e di 1 colonna a destra, a partire dal riferimento della cella correntemente puntata dal Per Ogni (nel nostro caso individuata dalla variabile C) e poi di nuovo di una colonna a destra. Le due righe vengono così lette come:

x è uguale al contenuto della cella posta una riga sopra e una colonna a destra rispetto a quella corrente;

la cella a destra di quella corrente è uguale al valore della cella corrente sommato ad x.

Contatori e Option Box

In figura 3 possiamo osservare il risultato finale: una finestra di dialogo appositamente creata per impostare la data nel formato, a scelta italiano o anglosassone, in una cella del foglio di lavoro. La realizzazione della dialog box è piuttosto semplice, e non trascende i metodi già illustrati nel precedente articolo. Ci limiteremo pertanto ad illustrare la presenza di due nuovi oggetti: le Option Box e le Caselle di Selezione.

I pulsanti di opzione sono raggruppati nella Casella di Gruppo a cui è stato dato il nome, non troppo fantasioso per la verità, «OPZIONI». In questo modo i due oggetti hanno funzione di scelta alternativa: se viene schiacciato il primo, il secondo si deseleziona automaticamente.

Le Caselle di Selezione, invece, hanno la caratteristica di poter essere personalizzate come valore massimo, minimo e avanzamento (step) utilizzando il comando Formato Oggetto da mouse (click con il pulsante destro sull'oggetto).

Diamo una breve occhiata al listato in figura 4: inizialmente, fuori dalle procedure, dichiariamo 3 variabili pubbliche, la procedura Incrementa() si occupa di visualizzare la Dialog Box e, in uscita da questa, di impostare nella cella B2 del foglio di lavoro la data scelta. Le funzioni Str ed ElimSpazi servono, rispettivamente a trasformare un numero in stringa e ad eliminare eventuali spazi residui.

Il Se... Allora... Altrimenti controlla se il primo Pulsante di Opzione è stato schiacciato o meno e di conseguenza riunisce le variabili GG, MM ed AA in ordine diverso a seconda che si desideri avere la data in formato anglosassone oppure no. Le procedure Incrementa_1, Incrementa_2 e Incrementa_3 sono collegate alle Caselle di Selezione e, attivate dall'uso stesso degli oggetti, riportano nelle variabili pubbliche G, M ed A i valori impostati dall'utilizzatore.

Si noti come in questo esempio il controllo dei valori impostati attraverso gli oggetti venga completamente svolto dalle procedure e non siano stati impostati collegamenti con le celle del foglio: la proprietà Celle, infatti, serve proprio a scrivere qualcosa in una delle celle del foglio.

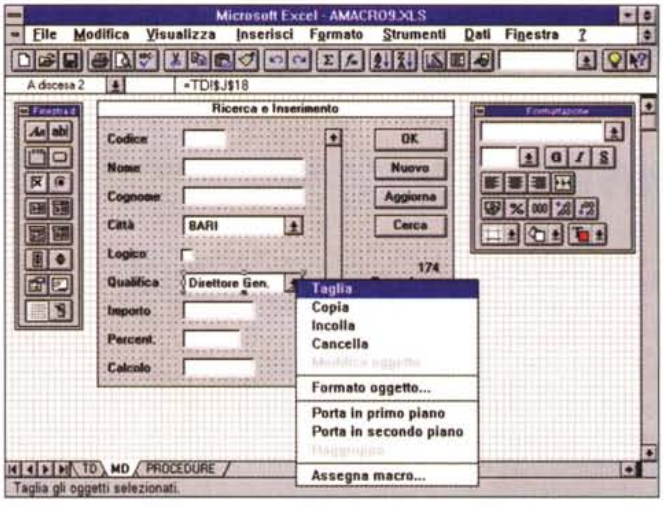

Figura 10 - Excel 5.0 - Visual Basic for Application - Il foglio nel quale si disegna la Dialog Box.
L'ambiente operativo è a quadretti (la quadrettatura obbliga a disegnare oggetti di dimensioni modulari) ed è caratterizzato dalla barra con una serie di strumenti che servono per definire i componenti della finestra di dialogo. Dovendo la box servire da interfaccia tra applicazione ed utente gli oggetti sono etichette, caselle di modifica, caselle di controllo, caselle di riepilogo, e così via.

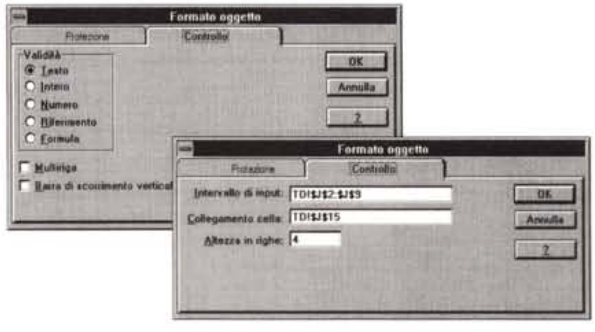

Figura 11 - Excel 5.0 - Visual Basic for Application - Una paio di finestre Formato oggetto.
Con un doppio click su un qualsiasi oggetto presente nella Dialog Box si apre la finestra Formato oggetto nella quale si impostano, oltre alle varie caratteristiche dipendenti dal tipo di oggetto, anche le regole del collegamento tra la box e il foglio con i dati.

Lettura di una directory

Nella figura 5 troviamo un esempio di procedura che combina Message Box standard con Dialog Box personalizzate e che, cosa forse più interessante, accede in modo mirato al disco. La procedura LeggiDir() in figura 6 si occupa di chiedere all'utente la directory di cui leggere il contenuto, vale a dire i nomi dei file. Propone poi una finestra di dialogo con una Casella di Riepilogo che permettere di scegliere dall'elenco proposto uno dei file per poi riportarlo nella cella A2 del foglio. La funzione FinestraInput serve a visualizzare la Message Box che, con il messaggio specificato nella variabile Msg, richiede all'utente la directory da sondare proponendone anche una di default, C:\WINDOWS.

CambiaDir imposta la directory per le operazioni future, mentre con la riga:

```
Nome = Dir(NomeDir)
```

impostiamo nella variabile il nome del primo file presente nella directory specificata precedentemente. Più avanti troveremo la semplice:

```
Nome = Dir()
```

che legge il nome di file successivo.

Nel loop controllato dall'istruzione Mentre... FineMentre (While... Wend, per i nostalgici), valido fintanto che il nome di file riportato dalla funzione Dir() ha lunghezza superiore a zero, usando la proprietà Elenco impostiamo i nomi di file, via via letti, nelle voci della Casella di Riepilogo della nostra Dialog Box. Si noti che precedentemente abbiamo azzerato il contenuto della Casella di Riepilogo con l'istruzione

```
FogliDialogo(1).CaselleDiRiepilogo(1).RimuoviTuttiElementi
```

la cui utilità si rivela nel caso in cui il numero di file presenti nella directory sia inferiore a quello della directory letta in precedenza: se non azzrassimo gli elementi della Casela di Riepilogo troveremmo, al termine dei file nuovi, la coda dei file della directory precedente. L'ultimo passaggio:

```
Celle(2; 1) = FogliDialogo(1).CaselleDiModifica(1).Testo
```

imposta nella cella A2 il nome di file selezionato nella Finestra di Dialogo.

In realtà, come in Visual Basic normale, esiste un metodo dedicato all'apertura di file che sfrutta le librerie di Windows per creare una classica finestra «Apri File» semplicemente specificando il tipo di file ed, eventualmente, le estensioni con i caratteri jolly:

```
Procedura Apri()
Nome = Applicazione.AttivaApriNomefile("Normale (*.xls), *.xls")
ApriFile nomefile:=Nome
Fine Procedura
```

```
Procedura Ricerca()
  ' Imposta i valori massimo e minimo della barra di scorrimento
  Con FogliDialogo("MD")
    .BarreDiScorrimento(1).Min = 2
    .BarreDiScorrimento(1).Max = Celle(12; 10) + 1
  .BarreDiScorrimento(1) = 1
  Scorri
  .Mostra
  Fine Con
Fine Procedura

Procedura Scorri()
  Con FogliDialogo("MD")
    Legge il valore impostato dalla Barra di Scorrimento per leggere dalla TD
    POS = .BarreDiScorrimento(1).Valore
    Imposta nella Finestra di Dialogo i valori letti nella TD e il calcolo
    .Etichette(11).Testo = Str(POS - 1)
    .Etichette(13).Testo = Str(Celle(12; 10))
    .CaselleDiModifica(1).Testo = Celle(POS; 1)
    .CaselleDiModifica(2).Testo = Celle(POS; 2)
    .CaselleDiModifica(3).Testo = Celle(POS; 3)
    .CaselleDiModifica(4).Testo = Celle(POS; 7)
    .CaselleDiModifica(5).Testo = Celle(POS; 8)
    .CaselleDiControllo(1).Valore = Celle(POS; 5)
    .CaselleDiModifica(6).Testo = Celle(POS; 7) * Celle(POS; 8)
  Fine Con
  ' Imposta due var. con i valori dei campi "CITTA" e "Q" del record corrente
  CITTA = Celle(POS; 4)
  QUAL = Celle(POS; 6)
  ' Trasforma la CITTA' del record corrente in un numero
  ' compreso fra 1 e 8 e lo imposta nella cella collegata
  ' con la Casella a Discesa 1, scandendo il contenuto delle
  ' celle appartenenti alla zona chiamata "Città", presente nel
  ' Foglio di Lavoro "TD":
  Contatore = 0
  Imposta Zona = Intervallo("Città")
  Per Ogni C In Zona
    Contatore = Contatore + 1
    Se C.Testo = CITTA Allora
      Esci Per
    Fine Se
  Successivo C
  Celle(15; 10).Valore = Contatore
  ' Trasforma la QUALIFICA del record corrente in un numero
  ' compreso fra 1 e 7 e lo imposta nella cella collegata
  ' con la Casella a Discesa 2, scandendo il contenuto delle
  ' celle appartenenti alla zona chiamata "Qualifiche", presente nel
  ' Foglio di Lavoro "TD"
  Contatore = 0
  Imposta Zona = Intervallo("Qualifiche")
  Per Ogni C In Zona
    Contatore = Contatore + 1
    Se C.Testo = QUAL Allora
      Esci Per
    Fine Se
  Successivo C
  Celle(18; 10).Valore = Contatore
Fine Procedura

Procedura Trova_Codice()
  ' Chiede il codice da ricercare
  CODICE = FinestraInput("Inserire il Codice", "Ricerca Record")
  Trovato = 0: Contatore = 0:Tot=Celle(12;10)+1
  Imposta Zona = Intervallo(Celle(2;1);Celle(Tot;1))
  Per Ogni C In Zona
    Contatore = Contatore + 1
    Se C.Testo = CODICE Allora
      Celle(21; 10) = Contatore + 1
      Trovato = 1
      Scorri
      Esci Per
    Fine Se
  Successivo C
  Se Trovato = 0 Allora
    Bip
    FinestraMessaggio "Record Non Presente in Archivio!", _
      vbEsclamazione, "ATTENZIONE!"
  Fine Se
Fine Procedura

Procedura Aggiorna()
 Con FogliDialogo("MD")
    R = Celle(21; 10) ' Riga corrente
    Celle(R; 1) = .CaselleDiModifica(1).Testo
    Celle(R; 2) = .CaselleDiModifica(2).Testo
    Celle(R; 3) = .CaselleDiModifica(3).Testo
    Celle(R; 7) = .CaselleDiModifica(4).Testo
    Celle(R; 8) = .CaselleDiModifica(5).Testo
    Se .CaselleDiControllo(1).Valore = -4146 Allora
      Celle(R; 5) = 0
    Altrimenti
      Celle(R; 5) = .CaselleDiControllo(1).Valore
    Fine Se
    CITTA = Celle(15; 10)
    QUAL = Celle(18; 10)
    Celle(R; 4) = Celle(CITTA + 1; 10)
    Celle(R; 6) = Celle(QUAL + 1; 13)
    .BarreDiScorrimento(1) = 1
    .BarreDiScorrimento(1).Max = R
  Fine Con
    Aggiorna la zona Database se è stato aggiunto un record
  FinestraMessaggio R
  Se R - 1 > Celle(12; 10) Allora
    N = "=R1C1:R" + ElimSpazioSx(Str(R)) + "C8"
    Nomi.Aggiungi Nome:="Database"; RiferitoAR1C1:=N
  Fine Se
  Scorri
Fine Procedura

Procedura Nuovo_record()  ' svuota le box
  Per C = 1 FinoA 6
    FogliDialogo("MD").CaselleDiModifica(C).Testo = ""
  Successivo
  FogliDialogo("MD").CaselleDiControllo(1).Valore = Falso
  Celle(15; 10) = 1: Celle(18; 10) = 1
  Celle(21; 10) = Celle(12; 10) + 2
Fine Procedura
```

Figura 12 - Excel 5.0 - Visual Basic for Application - I listati dell'ultimo esercizio.
L'operatività della nostra finestra di dialogo è gestita attraverso una serie di pulsanti e una barra di scorrimento, che, come si può desumere dal listato e dalla sua descrizione fatta nel testo, serve per andare avanti e indietro nei record della tabella con i dati. Le funzionalità attivabili dai pulsanti sono quelle standard in una maschera di gestione archivi.

Gestione elementare di un elenco

In figura 9 osserviamo l'ultimo esempio di questa puntata: la semplice gestione di un elenco di dati riportati in colonne.

Tenendo presente il «Modulo» fornito da Excel per la manipolazione degli archivi (menu Dati), abbiamo voluto crearne un clone sfruttando procedure VBA, collegamenti con il foglio di lavoro e formule di DataBase.

La Dialog Box è assai complessa (fig. 10): 6 Caselle di Modifica standard (Codice, Nome, Cognome, Importo, Percentuale, Calcolo) che verranno gestite da procedura, 2 Caselle a Discesa (Città e Qualifica), una Option Box (Logico), una Barra di Scorrimento, 4 pulsanti. L'utente può scorrere i record attraverso la Barra di Scorrimento, cercare un record tramite il numero di codice, inserire un nuovo record e aggiornare l'archivio con le modifiche effettuate al record corrente.

La celle J15 e J18 del foglio di lavoro sono collegate alle Caselle a Discesa e visualizzano sotto forma numerica la città e la qualifica scelte dall'utente. Verranno utilizzate nelle procedure per l'inserimento e la modifica dei dati dell'elenco attraverso la finestra di dialogo.

Nella cella J2 è impostata la funzione

```
=DB.CONTA.VALORI(Database;"COD";I1:I2)
```

che si occupa di contare i record presenti nel nostro archivio, a cui è stato dato il nome di zona convenzionale «DATABASE» (Inserisci/Nomi/Definisci).

La cella J21 è invece collegata con la Barra di Scorrimento e, in pratica, riporta il valore della riga corrente.

La zona J2:J9 e la zona M2:M8 sono state battezzate rispettivamente «CITTÀ» e «QUALIFICHE».

Analizziamo il flusso delle procedure

di controllo (fig. 12).

Al pulsante presente nel foglio di lavoro è collegata la procedura Ricerca() che si occupa di impostare i valori estremi della Barra di Scorrimento (attingendo al numero dei record calcolato nella cella J2), impostare ad 1 il record corrente, lanciare la procedura Scorri() e mostrare la finestra di dialogo.

Quando l'utente utilizza la Barra di Scorrimento attiva la procedura Scorri(). Questa, basandosi sul numero di record corrente (valore impostato con la Barra di Scorrimento), legge i dati dalle celle del foglio di lavoro e li riporta nelle Caselle di Modifica della Dialog Box. Inoltre aggiorna le informazioni per l'utente utilizzando due etichette per riportare il numero di record corrente e il numero totale di record.

Per i campi CITTÀ e QUALIFICA, il discorso è un po' più difficile: nelle celle, infatti, i dati appaiono in forma testuale, mentre per le Caselle a Discesa si tratta di valori numerici. La procedura utilizza perciò le due zone del foglio «CITTÀ» e «QUALIFICHE» per trasformare il contenuto (testuale) delle celle in valori numerici ed impostare questi nelle celle J15 e J18 collegate con le Caselle a Discesa:

```
Imposta Zona = Intervallo("Città")
Per Ogni C In Zona
...
Se C.Testo = CITTA Allora
Esci Per
Fine Se

Successivo
```

Da notare la presenza del metodo Intervallo riferito ad un range name del foglio e al comodo Esci Per che consente l'uscita dal Per Ogni ... Successivo in caso di conseguito risultato.

Il pulsante «Cerca» della Dialog Box attiva la procedura Trova_Codice(). Chiesto all'utente il numero di codice desiderato, questo viene ricercato nella zona di celle che va da A2 fino alla riga dell'ultimo record dell'archivio. In caso di insuccesso una FinestraMessaggio ed un segnale acustico (Bip) avvertono dell'errore.

In caso contrario, il puntatore viene mosso fino alla riga del record ricercato e il richiamo della procedura Scorri() serve ad aggiornare le caselle con i dati correnti.

La procedura Aggiorna(), connessa con l'omonimo pulsante, scarica i dati dalle Caselle di Modifica, Caselle a Discesa e Pulsante di Opzione nelle celle del record corrente o del nuovo record (qualora si sia preventivamente schiacciato il pulsante «Nuovo»).

Attenzione al Pulsante di Opzione CasellaDiControllo(1): in caso di azzeramento del valore può restituire un valore diverso da 1 o 0: -4146! Sull'argomento stiamo svolgendo accurate indagini e vi faremo sapere cosa significa tale stranezza. Un controllo in più ci consente comunque di evitare brutte sorprese.

Sempre in caso che sia stato aggiunto un ulteriore record, viene aggiornata la zona «DATABASE», in modo che la funzione DB.CONTA.VALORI nella cella J2 possa continuare a funzionare. Si noti la curiosa sintassi del metodo Nomi.Aggiungi:▼

```
N = "=R1C1:R" + ElimSpazioSx(Str(R)) + "C8"
Nomi.Aggiungi Nome:=«Database»; RiferitoAR1C1:=N
```

Per ultima la procedura richiamata dal pulsante «Nuovo».

Nuovo_record() si occupa di cancellare il contenuto da tutte le Caselle di Modifica, di resettare alla prima voce di elenco le Caselle a Discesa e di mettere a 0 la Option Box infine punta la barra di scorrimento (e con essa tutti i riferimenti) ad una riga nuova in fondo all'archivio.

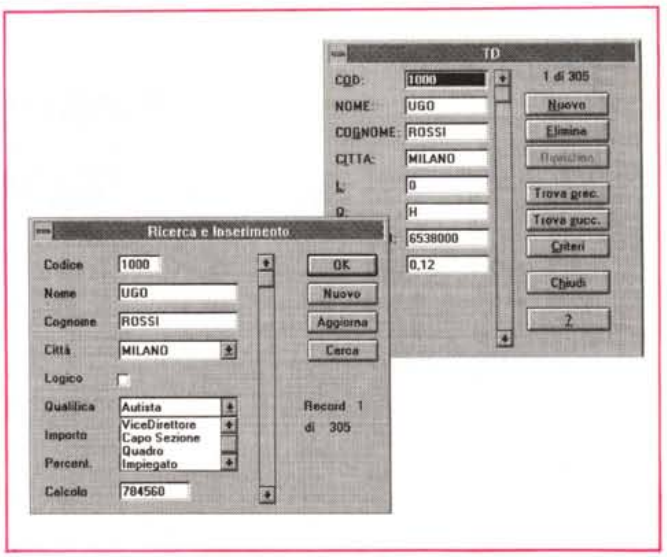

Figura - 13 Excel 5.0 - Visual Basic for Application - Due Finestre di Dialogo a confronto. Vediamo due finestre di dialogo, quella generata automaticamente con il comando Dati Modulo, e quella realizzata da noi. Le differenze tra le due sono sostanziali. Nella seconda abbiamo potuto inserire pulsanti a volontà, etichette a volontà, abbiamo potuto impostare differenti larghezze delle caselle di modifica, e via dicendo.

Alcune considerazioni già le possiamo fare. La prima è che VBA è più complesso del linguaggio Macro, sia perché le istruzioni usabili nel foglio Modulo non sono le stesse usabili nel foglio di Lavoro, sia perché le «parole chiave» sono molte di più.

Essendo più complesso permette soluzioni applicative meno legate all'ambiente Excel e più vicine all'ambiente Windows.

In questo VBA è più vicino al VB normale di quanto non lo sia al linguaggio macro di Excel.

Ad esempio il nostro esercizio «Lettura di una Directory» è sbagliato, in quanto VBA può richiamare direttamente, come detto, i servizi delle Common Dialog Box ed in particolare della File Apri, ben più sofisticata ed efficace di una semplice InputBox.

Un ulteriore esempio della vicinanza tra VBA e VB è costituito dalla gestione dell'evento Timer. In VB esiste il controllo Timer.

Con le poche righe che seguono è possibile, anche in VBA, impostare il lancio automatico ogni tre minuti della procedura Avviso():▼

```
Procedura MioTimer()
Applicazione.SuOra Adesso + OrarioVal("00.03.00"); "Avviso"
Fine Procedura
Procedura Avviso()
FinestraMessaggio "Ciao, mondo!"
MioTimer
Fine Procedura
```

Conclusioni

Siamo appena agli inizi della nostra manovra di avvicinamento al VBA.

Non mancherà occasione di tornare su questo e su altri argomenti. MC