Menu e riga di stato in Turbo Vision

di Sergio Polini (MC1166 su MC-link)

La volta scorsa abbiamo riflettuto sulla rapida evoluzione in atto nel mondo del software di sistema per PC, sottolineando l'importanza, in una fase di transizione, di tecniche e strumenti che consentano la scrittura di programmi facilmente portabili da un ambiente ad un altro. Abbiamo quindi evidenziato le principali analogie tra Turbo Vision e ObjectWindows. Non meno importanti sono tuttavia le differenze. Cominceremo quindi ora ad esaminare cosa c'è di diverso e come realizzare programmi sotto DOS e sotto Windows in modo tale da ridurre quanto più possibile il lavoro necessario per portare un'applicazione da un ambiente ad un altro. Quanto vedremo sarà comunque utile anche nel caso si sviluppi sotto uno solo dei due ambienti, in quanto avremo modo di mettere a punto tecniche per la realizzazione di menu «dinamici», per la visualizzazione di messaggi d'aiuto in una riga di stato, per l'implementazione di un sistema di help *contestuale, ecc.*

Non intendo proporvi un «corso» né di Turbo Vision né di programmazione sotto Windows con ObjectWindows. I manuali della Borland (nonostante qualche incertezza nell'edizione italiana del primo) sono molto ben fatti ed ampiamente sufficienti per un primo approccio all'uso delle due gerarchie di classi. Per portare avanti la nostra esplorazione, tuttavia, dovremo affrontare aspetti che, per quanto probabili in una vera applicazione, non hanno trovato adeguato spazio nei manuali dei compilatori. Basti pensare alle specifiche MDI. Molte applicazioni Windows (il *File Manager*, WinWord, Excel, lo stesso Turbo Pascal per Windows) sono in realtà applicazioni MDI, ed è ben probabile che tale debba essere qualsiasi applicazione che non sia una mera esercitazione o una semplice *utility*. Nonostante questo, i manuali Borland dedicano solo sei pagine alle applicazioni MDI; sono peraltro in buona compagnia: nella *Guide to Programming* dell'SDK si arriva a dieci, nel libro di Petzold le pagine di testo (esclusi cioè i listati) sono undici. Può quindi essere utile approfondire un po' l'argomento.

C'è un altro motivo per cui è consigliabile guardare più da vicino le specifiche MDI. In un'applicazione Turbo Vision, l'intero schermo del PC è occupato da un unico oggetto detto *DeskTop*, delimitato da un menu principale e da una riga di stato. In Windows siamo abituati ad uno scenario ben diverso: lo schermo viene presto popolato da diverse applicazioni, ognuna nella sua finestra, ed è possibile passare dall'una all'altra con un *click* del mouse. Si può, è vero, rendere la finestra di un'applicazione grande quanto l'intero schermo, ma è tutt'altro che immediata l'analogia tra una finestra «massimizzata» e il *DeskTop* del Turbo Vision: tanti tipi di finestre da un lato, uno stesso rigido e vuoto schema dall'altro. A ben vedere, tuttavia, un'applicazione Turbo Vision assomiglia non poco ad un'applicazione MDI; in entrambi i casi non si scrive nello spazio primario (l'area del *DeskTop* compresa tra il menu e la riga di stato, o

```
/* unmenu.rc */

#include "unmenu.h"

UnMenu MENU LOADONCALL MOVEABLE DISCARDABLE
BEGIN
  POPUP "&File"
  BEGIN
    MENUITEM "&Nuovo", IDM_NUOVO
    MENUITEM "&Apri ...", IDM_APRI
    MENUITEM "&Salva", IDM_SALVA
    MENUITEM "Salva con no&me ...", IDM_SALVA1
    MENUITEM SEPARATOR
    MENUITEM "S&tampa ...", IDM_STAMPA
    MENUITEM SEPARATOR
    MENUITEM "&Esci \tAlt+F4", IDM_ESCI
  END
  POPUP "&Modifica"
  BEGIN
    MENUITEM "&Taglia \tMaius+Canc", IDM_TAGLIA
    MENUITEM "&Copia \tCtrl+Ins", IDM_COPIA
    MENUITEM "&Aggiungi \tMaius+Ins", IDM_AGGIUNGI
    MENUITEM "Ca&ncella \tCanc", IDM_CANCELLA
  END
  POPUP "&Vista"
  BEGIN
    MENUITEM "&Totale", IDM_TOTALE
    MENUITEM "&Parziale", IDM_PARZIALE
  END
  POPUP "&Opzioni"
  BEGIN
    MENUITEM "&Riga di stato", IDM_RIGA
  END
  POPUP "Fines&tre"
  BEGIN
    MENUITEM "&Accosta", IDM_ACCOSTA
    MENUITEM "&Sovrapponi", IDM_SOVRAPPONI
    MENUITEM "&Chiudi tutto", IDM_CHIUDI
  END
  POPUP "&Guida"
  BEGIN
    MENUITEM "&Usare la guida", IDM_USARE
    MENUITEM "&Guida estesa", IDM_GUIDA
    MENUITEM "&Tastiera", IDM_TASTIERA
    MENUITEM "&Indice", IDM_INDICE
    MENUITEM "&Nozioni di base", IDM_NOZIONI
    MENUITEM SEPARATOR
    MENUITEM "Inform&azioni", IDM_INFORMAZIONI
  END
END
```

Figura 1 - Un esempio di definizione di un menu mediante uno script file *da compilare con RC.EXE. Ometto il file UNMENU.H, in quanto contiene solo la definizione delle costanti simboliche, come IDM_NUOVO, IDM_APRI, ecc.*

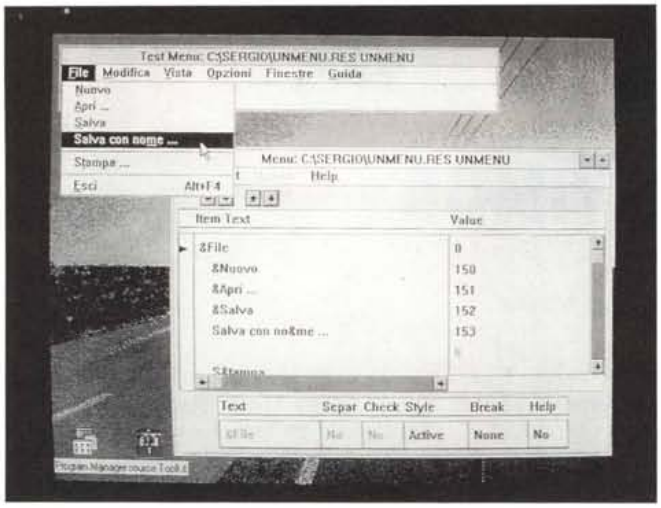

Figura 2 - Costruzione interattiva di un menu con il Whitewater Resource Toolkit. La finestra superiore mostra <lo stato dei lavori> mentre si procede nella definizione dei diversi livelli del menu.

la *client area* della *main Window*), ma si aprono uno o più spazi di lavoro (finestre) che possiamo muovere, allargare, restringere, accostare, sovrapporre mediante il semplice ricorso a funzionalità incorporate.

Vi sono certo delle differenze: un *DeskTop* ha praticamente sempre una riga di stato, che non fa parte delle funzionalità già incorporate nella MDI di Windows; un'applicazione MDI ha sempre un menu dedicato alla manipolazione delle finestre, che è invece opzionale in Turbo Vision; si tratta soprattutto di un menu che si allunga e restringe automaticamente, riportando costantemente i nomi delle finestre aperte, ed ha quindi un comportamento un po' diverso dai menu del Turbo Vision.

E così via. Aggiungere una riga di stato ad un'applicazione MDI non è certo impossibile, come testimoniano le applicazioni che prima ricordavo; sarà quindi utile vedere come si deve fare. Analogamente si possono manipolare i menu del Turbo Vision, e vedere come si deve fare può essere utile sia per facilitare il *porting* di un'applicazione sotto Windows, sia per arricchire comunque le nostre applicazioni sotto DOS. Potremmo anche considerare che Windows si trova installato su un enorme numero di macchine e che, per quanto non tutti lo usino costantemente, dare la stessa apparenza ad applicazioni DOS e Windows non può che facilitare le cose agli utenti.

La risorsa <menu>

Vediamo quindi come si costruisce un menu con il Turbo Vision, per poi in seguito modificarne la struttura.

Il nostro obiettivo, lo ripeto, è quello di scrivere applicazioni Turbo Vision e ObjectWindows che si assomiglino quanto più possibile, e di farlo in modo che siano facilmente portabili dall'uno all'altro ambiente. Per fare questo, si deve ricordare che, programmando sotto Windows, si fa ampio uso delle *risorse*: non solo icone e bitmap, ma anche i menu e insieme di stringhe (messaggi d'errore, ecc.) sono risorse, cioè qualcosa che viene costruito fuori dal programma e poi a questo aggiunto, in modo tale che è possibile cambiare la risorsa senza dover intervenire sul programma. Si possono così cambiare le descrizioni delle opzioni di un menu, i messaggi d'errore, i messaggi che compaiono sulla riga di stato, senza alcuna modifica ai sorgenti.

Nei primi tempi della programmazione sotto Windows, menu e stringhe venivano definiti mediante file di testo di apposito formato (figura 1), che venivano poi trattati con RC.EXE, il *resource compiler* della Microsoft. Ancora oggi può essere conveniente, in alcuni casi, usare RC.EXE (non a caso fornito insieme al Turbo Pascal per Windows), ma si preferisce usare strumenti come il Whitewater Resource Toolkit o il recen-

```
Program BuildRes;

uses Objects, Drivers, Views, Menus, App;

const
                                hcFile          = 2002;
  cm_MDIFileNew    = 151;       hc_MDIFileNew   = 1000 + cm_MDIFileNew;
  cm_MDIFileOpen   = 152;       hc_MDIFileOpen  = 1000 + cm_MDIFileOpen;
  cm_FileSave      = 153;       hc_FileSave     = 1000 + cm_FileSave;
  cm_FileSaveAs    = 154;       hc_FileSaveAs   = 1000 + cm_FileSaveAs;
  cm_FilePrint     = 160;       hc_FilePrint    = 1000 + cm_FilePrint;
                                hcQuit          = 1000 + cmQuit;

var
  ResFile: TResourceFile;

procedure CreateMenuBar;
var
  M: PMenuBar;
  R: TRect;
begin
  RegisterMenus;
  R.Assign(0, 0, 80, 1);
  M := New(PMenuBar, Init(R, NewMenu(
    NewSubMenu('~F~ile', hcFile, NewMenu(
      NewItem('~N~uovo', '', kbNoKey, cm_MDIFileNew, hc_MDIFileNew,
      NewItem('~A~pri ...', '', kbNoKey, cm_MDIFileOpen, hc_MDIFileOpen,
      NewItem('~S~alva', '', kbNoKey, cm_FileSave, hc_FileSave,
      NewItem('Salva con no~m~e ...', '', kbNoKey, cm_FileSaveAs,
              hc_FileSaveAs,
      NewItem('S~t~ampa ...', '', kbNoKey, cm_FilePrint, hc_FilePrint,
      NewItem('~E~sci', 'Alt+F4', kbAltF4, cmQuit, hcQuit,
      nil)))))),
    nil))
  ));
  ResFile.Put(M, 'RESDEMO_MB');
  Dispose(M, Done);
end;

begin
  ResFile.Init(New(PBufStream, Init('RESDEMO.RES', stCreate, 1024)));
  CreateMenuBar;
  ResFile.Done;
end.
```

Figura 3 - Un programma BUILDRES che costruisce il sistema di menu di un'applicazione e lo salva in un file RESDEMO.RES.

te Resource Workshop della Borland, che consentono di costruire le risorse interattivamente su video (figura 2).

Anche nel Turbo Vision si possono definire risorse fuori del programma, ma per fare questo occorre scrivere un altro programma, come illustrato nel capitolo 9 della *Turbo Vision Guide* ed esemplificato dai vari GENRDEMO e TVRDEMO forniti insieme al compilatore. Nelle figure 3 e 4 vi propongo un ulteriore esempio di definizione di risorse: un programma BUILDRES costruisce un semplice menu e lo salva in un file RESDEMO.RES; il programma RESDEMO si appropria del file di risorse, venendo così a disporre del menu senza bisogno di costruirselo.

Quest'ultimo punto merita un chiarimento. Uno degli aspetti più comodi della programmazione per eventi è rappresentato dal fatto che gli eventi per i quali non sia prevista una specifica risposta vengono semplicemente ignorati. Definendo un sistema di menu, associamo ad ogni opzione un comando: quando l'utente sceglierà quell'opzione, con il mouse o con la tastiera, verrà generato un evento di tipo *evCommand* con un valore uguale a quello del numero identificativo del comando associato all'opzione. Il Turbo Vision già sa come rispondere ad alcuni comandi predefiniti come *cmQuit*, che vale 1 e che provoca la fine del programma e il ritorno al DOS. Per i comandi definiti dall'utente non esistono, ovviamente, risposte predefinite, ma questo non comporta alcun problema: non si genera una situazione di errore, semplicemente non succede nulla. Questo consente di prevedere un gran numero di eventi, implementando un po' alla volta, con metodi di una classe derivata da *TApplication*, la risposta che l'applicazione deve dare ad ognuno di essi; è ad esempio possibile costruire un prototipo di un'applicazione con tutto il suo complesso e articolato sistema di menu, per avere sin dall'inizio un'idea concreta di quello che sarà il prodotto finito, procedendo poi con calma e gradualità fino al completamento di tutti i dettagli. Nel nostro caso, si potrebbe estendere il menu in BUILDRES senza modificarne in alcun modo RESDEMO.EXE; se poi durante il completamento di RESDEMO ci accorgessimo che il sistema di menu richiede qualche ritocco, potremmo subito intervenire su BUILDRES per avere un'idea aggiornata del programma nella sua veste definitiva.

La risorsa <stringhe>

In futuro avremo occasione di comportarci proprio in questo modo. Non vi sarà del resto sfuggito un altro indizio della mia preoccupazione di preparare sin da questi semplici esempi i futuri sviluppi del nostro discorso: alcuni comandi hanno nomi identici a quelli definiti in ObjectWindows per l'interfaccia MDI, prevedo l'uscita dal programma con Alt+F4 invece che con Alt+X in ossequio alle specifiche SAA-CUA.

Ora tuttavia dobbiamo arricchire un po' il nostro RESDEMO; si desidera spesso, infatti, proporre all'utente che percorra le opzioni di un menu un messaggio esplicativo di ognuna nella riga di stato. Data la stretta parentela tra tali messaggi e le opzioni del sistema dei menu, è chiaramente preferibile realizzare anche questi come risorse. Il Turbo Vision propone le classi *TStrListMaker* per la costruzione di liste di stringhe e *TStringList* per il loro uso, che ben si prestano allo scopo; purtroppo però né i manuali né (salvo mia svista) i numerosi *demo* ci offrono esempi concreti al riguardo. Soprattutto, può sfuggire la necessità di derivare una nuova classe da *TStatusLine*, allo scopo di ridefinire il metodo *Hint* di questa.

Procediamo con ordine. Notiamo innanzitutto che nella figura 3 le definizioni delle costanti relative ai comandi, il cui nome comincia con *cm*, sono affiancate da costanti il cui nome comincia con *hc*; si tratta di costanti relative all'*help contestuale*, anch'esse, come i comandi, associate alle singole opzioni del menu (i valori di tali costanti non devono essere inferiori a 1000, in quanto il range 0..999 è riservato). La classe *TStatusLine* dispone di parte della funzionalità necessaria per visualizzare un messaggio sulla riga di stato: ogni volta che il *focus* passa da una vista ad

```
Program ResDemo;

uses Objects, Drivers, Views, Menus, App;

var
  ResFile: TResourceFile;

type
  TMyApp = object(TApplication)
    constructor Init;
    procedure InitMenuBar; virtual;
    procedure InitStatusLine; virtual;
  end;

constructor TMyApp.Init;
begin
  ResFile.Init(New(PBufStream, Init('RESDEMO.RES', stOpenRead, 1024)));
  RegisterMenus;
  TApplication.Init;
end;

procedure TMyApp.InitMenuBar;
begin
  MenuBar := PMenuBar(ResFile.Get('RESDEMO_MB'));
end;

procedure TMyApp.InitStatusLine;
var
  R: TRect;
begin
  GetExtent(R);
  R.A.Y := R.B.Y - 1;
  StatusLine := New(PStatusLine, Init(R,
    NewStatusDef(0, $FFFF,
      NewStatusKey('', kbF10, cmMenu,
      NewStatusKey('~Alt+F4~ Esci', kbAltF4, cmQuit,
      nil)),
    nil))
  );
end;

var
  MyApp: TMyApp;

begin
  MyApp.Init;
  MyApp.Run;
  MyApp.Done;
end.
```

Figura 4 - Esempio di uso di un file di risorse in un'applicazione Turbo Vision.

un'altra (dove per vista si intende qualsiasi oggetto visibile sullo schermo: una finestra, una check box, un pulsante, una scrollbar, ecc.), viene mandato alla riga di stato il messaggio <hint> (che potremmo tradurre: dai un suggerimento all'utente) con un argomento pari al valore della costante di help associata a quella vista; la riga di stato risponde con il suo metodo *Hint*, che però ritorna una stringa nulla. Il programmatore è lasciato libero di derivare una sua classe da *TStatusLine* e di ridefinire il metodo *Hint* come meglio crede, ad esempio scrivendo una lunga istruzione **case**. Né comodo né elegante; meglio un qualcosa che ci offra subito la stringa corrispondente ad un dato valore della costante di help.

Ecco l'utilità della classe *TStringList*. Per dirla alla Smalltalk, si tratta di un dizionario, cioè di un insieme di associazioni <chiave, stringa>, dove la chiave è un numero compreso tra 0 e 65535; sono presenti solo un constructor che carica la lista di stringhe da un file di risorse, un destructor, e un metodo *Get* che fa proprio al caso nostro: richiede un argomento di tipo *word* e ritorna la stringa la cui chiave è uguale all'argomento. Potete trovare nella figura 5 una versione di RESDEMO in cui vengono tratti da un file di risorse sia il sistema di menu che le descrizioni di ogni opzione di questo destinate ad apparire sulla riga di stato. Da notare la derivazione di una nuova classe da *TStatusLine* e la ridefinizione del metodo *TApplication.InitStatusLine*: viene prevista, prima della riga di stato valida per tutti i valori delle costanti di help (da 0 a SFFFF), una riga di stato più corta per i valori da 1000 a 3000; in occasione di tali valori, la riga di stato verrà infatti allungata a destra con una barra verticale seguita dalla stringa ritornata da *Hint*.

Un'istanza di *TStringList* può leggere dal file solo liste di stringhe preparate con un'istanza di *TStrListMaker*. Dei quattro metodi di essa ci interessa soprattutto *Put*, che aggiunge alla lista coppie <chiave, stringa>. Per modificare BUILDRES in modo da integrare il file di risorse con una lista di stringhe, basta definire una funzione *CreateStringList* e chiamarla lì dove viene chiamata anche *CreateMenuBar* (figura 6).

```
Program ResDemo;

uses Objects, Drivers, Views, Menus, App;

var
  ResFile: TResourceFile;
  HintList: PStringList;

type
  PMyStatusLine = ^TMyStatusLine;
  TMyStatusLine = object(TStatusLine)
    constructor Init(var Bounds: TRect; ADefs: PStatusDef);
    function Hint(AHelpCtx: Word): String; virtual;
  end;
  TMyApp = object(TApplication)
    constructor Init;
    procedure InitMenuBar; virtual;
    procedure InitStatusLine; virtual;
  end;

constructor TMyStatusLine.Init(var Bounds: TRect; ADefs: PStatusDef);
begin
  HintList := PStringList(ResFile.Get('RESDEMO_HT'));
  TStatusLine.Init(Bounds, ADefs);
end;

function TMyStatusLine.Hint(AHelpCtx: Word): String;
begin
  if HintList <> nil then
    Hint := HintList^.Get(AHelpCtx)
  else
    Hint := '';
end;

constructor TMyApp.Init;
begin
  ResFile.Init(New(PBufStream, Init('RESDEMO.RES', stOpenRead, 1024)));
  RegisterMenus;
  RegisterType(RStringList);
  TApplication.Init;
end;

procedure TMyApp.InitMenuBar;
begin
  MenuBar := PMenuBar(ResFile.Get('RESDEMO_MB'));
end;

procedure TMyApp.InitStatusLine;
var

  R: TRect;
begin
  GetExtent(R);
  R.A.Y := R.B.Y - 1;
  StatusLine := New(PMyStatusLine, Init(R,
    NewStatusDef(1000, 3000,
      NewStatusKey('~F1~ Guida', kbNoKey, 0,
      nil),
    NewStatusDef(0, $FFFF,
      NewStatusKey('', kbF10, cmMenu,
      NewStatusKey('~Alt+F4~ Esci', kbAltF4, cmQuit,
      nil)),
    nil))
  ));
end;

var
  MyApp: TMyApp;

begin
  MyApp.Init;
  MyApp.Run;
  MyApp.Done;
end.
```

Figura 5 - Una nuova versione di RESDEMO; questa volta vengono tratti dal file di risorse sia il menu che le descrizioni delle opzioni di questo destinate ad apparire nella riga di stato.

Un resource editor

Possiamo intervenire su BUILDRES per allargare il sistema di menu o cambiarne l'articolazione, per aggiornare le descrizioni delle diverse opzioni, senza per questo dover modificare in alcun modo (neppure ricompilare) RESDEMO. Però occorre modificare BUILDRES, e quindi il beneficio è solo apparente. Sotto Windows tutto è reso più comodo dai *resource editor*: invece di riprogrammare, si <ridisegna> il nuovo menu: più semplice, più comodo, più veloce ed anche più sicuro, in quanto si può tenere costantemente sotto controllo visivo lo stato dei lavori; è infatti disponibile una finestra in cui poter verificare la nuova struttura del menu, anche percorrendo e scegliendo le varie opzioni, come se si trattasse del >vero< menu dell'applicazione.

Come vi dicevo già il mese scorso, la Blaise Computing propone un analogo strumento per il Turbo Vision. Il suo Turbo Vision Development Toolkit comprende infatti un programma RESEDIT che, come si indovina dal nome, è un vero e proprio editor di risorse paragonabile, nel modo carattere, a quelli usabili sotto Windows. La figura 7 mostra RESEDIT mentre si sta definendo un menu un po' più complesso di quello esemplificato nei programmi appena visti, e sulla cui struttura avremo modo di tornare.

Devo dire che RESEDIT non è perfetto. Lo uso regolarmente da mesi, e qualche volta mi ha sparato un poco gradevole *run-time error*: una sola volta, tuttavia, ciò ha comportato la perdita del lavoro svolto. In generale, lo trovo uno strumento molto comodo nonostante qualche pecca, e di cui ormai non so fare a meno. Non vi dico questo per spingervi ad acquistarlo, ma per chiarire i motivi per cui, in seguito, non riprodurrò più listati contenenti la definizione di menu, dialog box o liste di stringhe: i manuali e i demo del Turbo Vision spiegano bene come costruire un menu o una dialog box, abbiamo visto sopra come costruire e usare liste di stringhe. Sarebbe quindi inutile ripetere linee e linee di sorgente (con tutte quelle parentesi!) per costruire risorse per le quali sarà sufficiente, ed anche più chiara, la riproduzione di una schermata. Per il resto, se avessi costruito con RESEDIT le risorse di RESDEMO, avrei dovuto apportare una sola modifica al programma: avrei dovuto dare estensione BRS invece che RES al file di risorse.

C'è anche un motivo di carattere banalmente pratico. La definizione di una risorsa da programma comporta a volte la scrittura di decine di righe di codice, che occuperebbero troppo del poco spazio che abbiamo a disposizione per parlare sia di Turbo Vision che di Object-Windows. Se limito ad una riga (con un *ResFile.Get*) il tutto, avrò più spazio per la trattazione discorsiva degli argomenti.

C'è infine un ultimo motivo. Il Turbo Vision Development Toolkit comprende anche un programma RC-1.EXE, definito dalla Blaise l'<inverso> di RC.EXE, il *resource compiler* della Microsoft: partendo da un file BRS, è in grado di produrre file da cui RC.EXE può derivare risorse direttamente usabili da applicazioni Windows. Contribuendo così ad accelerare la conversione da un ambiente all'altro (il file della figura 1 è stato ottenuto proprio in questo modo, partendo dal menu che potete vedere in corso di costruzione nella figura 7; ho solo tolto la sezione relativa al *system menu*, automaticamente disponibile sotto Windows).

```
...

procedure CreateStringList;
var
  S: TStrListMaker;
begin
  RegisterType(RStrListMaker);
  S.Init(16384, 256);
  S.Put(hcFile, 'Menù File');
  S.Put(hc_MDIFileNew, 'File/Nuovo');
  S.Put(hc_MDIFileOpen, 'File/Apri');
  S.Put(hc_FileSave, 'File/Salva');
  S.Put(hc_FileSaveAs, 'File/Salva con nome');
  S.Put(hc_FilePrint, 'File/Stampa');
  S.Put(hcQuit, 'File/Esci');
  ResFile.Put(@S, 'RESDEMO_HT');
  S.Done;
end;

...

begin
  ResFile.Init(New(PBufStream, Init('RESDEMO.RES', stCreate, 1024)));
  CreateMenuBar;
  CreateStringList;
  ResFile.Done;
end.
```

Figura 6 - Le modifiche da apportare a BUILDRES per aggiungere al file di risorse anche una stringa di liste.

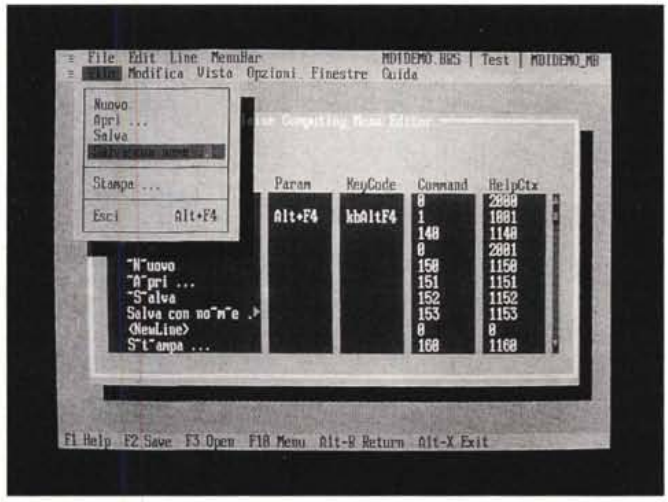

Figura 7 - Il RESEDIT della Blaise Computing al lavoro su un menu. Premendo Alt+M si può avere in ogni momento un'idea di quello che sarà il menu dell'applicazione.

La conversione non è sempre perfetta, in quanto rimangono alcune differenze (le check box, ad esempio, funzionano in maniera un po' diversa in Turbo Vision e in Windows: nel primo fanno sempre <gruppo>, nel secondo bisogna dichiarare esplicitamente un GROUPBOX), ma soprattutto perché vi sono state poche ma noiose disattenzioni nel disegno di RC-1.EXE; tanto per fare un esempio, se indico con <Apri...> l'opzione di un menu, RC-1 dà il nome di <IDM_APRI...> (compresi i puntini) alla corrispondente costante simbolica, con il risultato di far arrabbiare RC.EXE. Basta in realtà mettere uno spazio tra <April> e i puntini, ma, dal momento che anche questi fanno in un certo senso parte dello standard, non sarebbe stato difficile prevedere un qualche automatismo. In alternativa, si può intervenire direttamente sui file prodotti da RC-1 e modificarli, come a volte è comunque necessario fare. A mio parere ne vale la pena, piuttosto che ricostruire tutto da capo per Windows, ma capirei bene un parere contrario. Fate voi.

Per ora, visti i già detti problemi di spazio, ci fermiamo qui. La volta prossima vedremo come fare per aggiungere una riga di stato ad un'applicazione MDI sotto Windows. MC