# Da un demo MDI alla storia di un bug

*di Sergio Polini*

*È giunto il momento di vedere in azione le nostre unit per applicazioni MDI, verificando quanto la OOP consenta di ridurre il lavoro necessario per adattare di volta in volta uno stesso schema a diverse situazioni. Non ci dimentichiamo tuttavia né del Turbo Vision né di tecniche più tradizionali. Torneremo il mese prossimo al Turbo Vision; già ora, tuttavia, vi parlerò un po' del suo* help compiler *(TVHC), per raccontarvi di un bug scoperto da un abbonato a MC-link e delle soluzioni che sono state lì proposte*

Vi propongo subito nelle figure 1 e 2 l'output del programma MDIDEMO. Nella prima vediamo cosa appare sullo schermo all'inizio: un menu ridotto all'essenziale, un ribbon con quattro pulsanti disabilitati, una *client window* vuota, una riga di stato già in grado di offrire aiuto sulle opzioni dei menu. Nella seconda schermata vediamo la *client window* popolata da alcune *child window* e il più ricco menu associato a quella attiva; i pulsanti del ribbon sono abilitati, le opzioni del menu Opzioni sono lì pronte per consentirci di far sparire, se così vogliamo, il ribbon e/o la riga di stato, come ci ricorda il messaggio che appare su quest'ultima.

Le finestre non fanno null'altro che confermarci che si tratta di *child window*, e ben poco succede se premiamo i pulsanti del ribbon (come potete indovinare scorrendo il listato riprodotto nella figura 3). Lo scopo dell'applicazione, infatti, è solo quello di mostrare come, con poche righe di codice, si possa realizzare una struttura complessa, i cui dettagli rimangono nascosti sia nei meccanismi di Windows, sia nelle unit che abbiamo messo a punto nei mesi scorsi. Struttura complessa, ma anche flessibile: non c'è alcun vincolo sulla struttura del ribbon, ad esempio, in quanto basta che si tratti di una dialog box non modale; le finestre appartengono ad un tipo *TChildWin* di grande semplicità, ma potrebbero ben essere quadri di un foglio elettronico o viste su un data base; la riga di stato, come già precisato a suo tempo, potrebbe avere più «campi».

## Uso delle unit MDI

Seguendo il listato di MDIDEMO (figura 3) vediamo, dopo la dichiarazione delle costanti per i pulsanti del ribbon e del consueto tipo *TMyApp*, quella per il ribbon dell'applicazione; qui compaiono solo le procedure che verranno eseguite alla pressione dei pulsanti. Altrettanto semplici le dichiarazioni delle classi per la *frame window* e le *child window*. Per quanto riguarda queste ultime, resta inteso che in una vera applicazione si avrebbe bisogno di ben altra complessità, ma ad una classe come *TFra-*

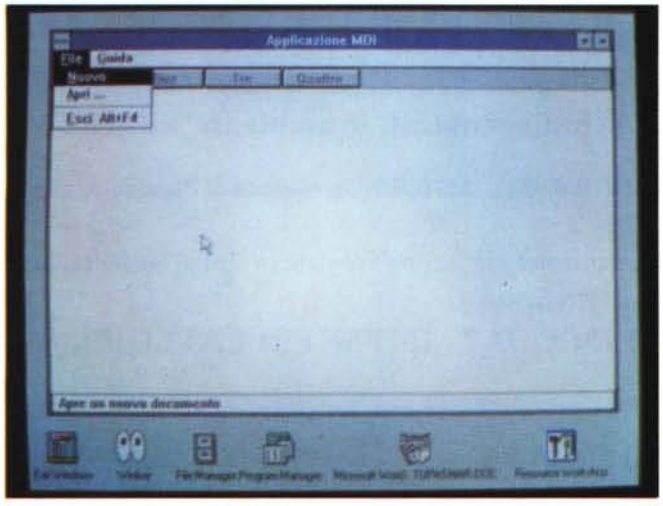



*Figura 1 - Il programma MDIDEMO appena lanciato.*

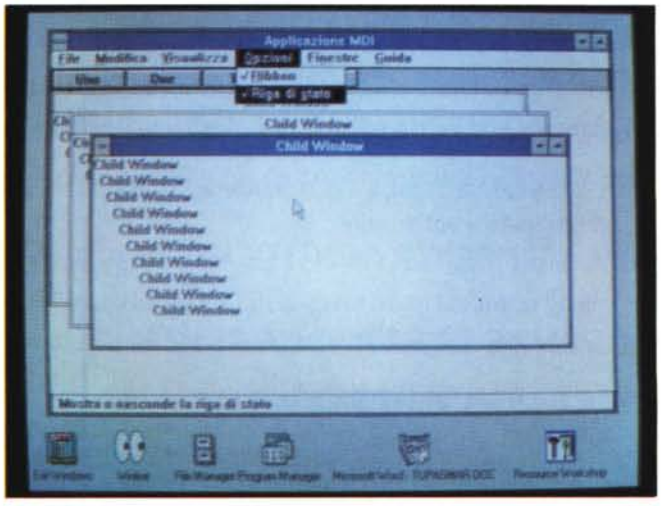



*Figura 2 - Il programma MDIDEMO dopo l'apertura di alcune* child window.

*meWin* rimarranno comunque assegnati solo gli stessi compiti: inizializzare il ribbon e la riga di stato nel constructor, aprire le singole finestre (qui con il metodo *InitChild*).

Il codice mi pare chiaro al punto da non richiedere particolari commenti. Voglio solo sottolineare che, perché lo schema generale funzioni, occorre che ad ogni «elemento d'interfaccia» previsto nel ribbon venga associato un «oggetto d'interfaccia» mediante un *InitResource*, secondo quel meccanismo su cui mi ero soffermato nella puntata di gennaio. In caso contrario, il metodo *WMEnable* del ribbon, visto sempre a gennaio, non riuscirebbe ad abilitare e disabilitare correttamente tutti gli «elementi» (avrei potuto rendere più generale il tutto implementando quel metodo con funzioni della API di Windows, ma mi pare che l'associazione oggetto-elemento mediante *InitResource* presenti vantaggi tali da consigliarne comunque l'uso; magari ne riparleremo).

È forse più delicata (o almeno più noiosa) la preparazione del file di risorse. Ho preferito evitare, per ovvie ragioni di spazio, la riproduzione dello script equivalente a quel MDIDEMO.RES che ho preparato con il Resource Workshop; vi propongo quindi in figura 4 uno schema che mostra l'essenziale, ovvero l'associazione delle opportune costanti di help (per i messaggi della riga di stato) ai vari menu e alle rispettive opzioni.

Nella prima colonna abbiamo, nell'ordine, il system menu dell'applicazione, quello delle *child window*, i vari menu della *frame window* e delle *child window*. Qesti ultimi hanno alcune parti comuni, riconoscibili dalla presenza di un doppio numero nella seconda colonna (ad esempio, l'opzione File/Nuovo compare sia nel menu della *frame window* che in quello delle *child window*; nella seconda colonna troviamo «2000/4000», ad indicare che nel primo caso si usa 2000, nel secondo 4000).

La seconda colonna riporta le costanti che i metodi *WMMenuSelect* del mese

```
program MDIDemo;
(*$R MDIDEMO.RES*)
uses
  WinTypes, WinProcs, WObjects,
  Stripes, MDIClnt, MDIChld, MDIFrame;
const
  id_Uno     = 101;
  id_Due     = 102;
  id_Tre     = 103;
  id_Quattro = 104;
type
  TMDIApp = object(TApplication)
    procedure InitMainWindow; virtual;
  end;
  PRibbon = ^TRibbon;
  TRibbon = object(TMDIRibbon)
    procedure IDUno(var Msg: TMessage); virtual id_First + id_Uno;
    (* procedure per altri pulsanti omesse per brevita' *)
  end;
  PFrameWin = ^TFrameWin;
  TFrameWin = object(TNewMDIFrame)
    constructor Init(ATitle: PChar; AMenu: HMenu);
    function InitChild: PWindowsObject; virtual;
  end;
  PChildWin = ^TChildWin;
  TChildWin = object(TNewMDIChild)
    procedure Paint(PaintDC: HDC; var PaintInfo: TPaintStruct); virtual;
  end;
procedure TMDIApp.InitMainWindow;
begin
  MainWindow := New(PFrameWin,
    Init('Applicazione MDI', LoadMenu(HInstance, 'FrameMenu')));
end;
constructor TFrameWin.Init(ATitle: PChar; AMenu: HMenu);
var
  i: Integer;
  B: PButton;
begin
  TNewMDIFrame.Init(ATitle, AMenu);
  Ribbon := New(PRibbon, Init(@Self, 'RIBBON', True));
  for i := id_Uno to id_Quattro do
    B := New(PButton, InitResource(Ribbon, i));
  StatusBar := New(PMDIStatusBar, Init(@Self, 'STATUSBAR', True));
  SBText := 'Frame Window';
end;
function TFrameWin.InitChild: PWindowsObject;
begin
  InitChild := New(PChildWin, Init(@Self, 'Child Window', 'ChildMenu'));
end;

procedure TRibbon.IDUno(var Msg: TMessage);
var
  Focused: HWnd;
  DC: HDC;
begin
  MessageBox(HWindow, 'Premuto!', 'Pulsante "Uno"',
    mb_IconExclamation or mb_Ok);
  (* Verifica che il 'focus' rimanga alla finestra attiva *)
  Focused := GetFocus;
  DC := GetDC(Focused);
  TextOut(DC, 0, 0, 'Premuto il pulsante "Uno".', 26);
  ReleaseDC(Focused, DC);
end;
procedure TChildWin.Paint(PaintDC: HDC; var PaintInfo: TPaintStruct);
var
  x, y, i: Integer;
  TM: TTextMetric;
begin
  GetTextMetrics(PaintDC, TM);
  x := 0;
  y := 0;
  for i := 1 to 10 do begin
    TextOut(PaintDC, x, y, 'Child Window', 12);
    Inc(x, TM.tmAveCharWidth);
    Inc(y, TM.tmHeight + TM.tmExternalLeading);
  end;
end;
var
  MDIApp: TMDIApp;
begin
  MDIApp.Init('MDIApp');
  MDIApp.Run;
  MDIApp.Done;
end.
```

*Figura 3 - Il listato del programma MDIDEMO.*

| Menu/Opzione | Base | Identificativo | Costante di help |
|---|---|---|---|
| System menu applicaz. | 1000 | 0 | 1000 |
| Ripristina | 2000 | $F120 -> 18 | 2018 |
| Muovi | 2000 | $F010 -> 1 | 2001 |
| Ridimensiona | 2000 | $F000 -> 0 | 2000 |
| Riduci a icona | 2000 | $F020 -> 2 | 2002 |
| Ingrandisci | 2000 | $F030 -> 3 | 2003 |
| Chiudi | 2000 | $F060 -> 6 | 2006 |
| Passa a... | 2000 | $F130 -> 19 | 2019 |
| System menu child | 3000 | 0 | 3000 |
| Ripristina | 4000 | $F120 -> 18 | 4018 |
| Muovi | 4000 | $F010 -> 1 | 4001 |
| Ridimensiona | 4000 | $F000 -> 0 | 4000 |
| Riduci a icona | 4000 | $F020 -> 2 | 4002 |
| Ingrandisci | 4000 | $F030 -> 3 | 4003 |
| Chiudi | 4000 | $F060 -> 6 | 4006 |
| Successiva | 4000 | $F040 -> 4 | 4004 |
| File | 1000/3000 | 1 | 1001/3001 |
| Nuovo | 2000/4000 | 24339 | 26339/28339 |
| Apri | 2000/4000 | 24332 | 26332/28332 |
| Salva | 4000 | 24333 | 28333 |
| Salva con nome | 4000 | 24334 | 28334 |
| Stampa | 4000 | 160 | 4160 |
| Esci | 2000/4000 | 24340 | 26340/28340 |
| Modifica | 3000 | 2 | 3002 |
| Taglia | 4000 | 24320 | 28320 |
| Copia | 4000 | 24321 | 28321 |
| Aggiungi | 4000 | 24322 | 28322 |
| Cancella | 4000 | 24323 | 28323 |
| Visualizza | 3000 | 3 | 3003 |
| Totale | 4000 | 180 | 4180 |
| Parziale | 4000 | 181 | 4181 |
| Opzioni | 3000 | 4 | 3004 |
| Ribbon | 4000 | 24316 | 28316 |
| Riga di stato | 4000 | 24317 | 28317 |
| Finestre | 3000 | 5 | 3005 |
| Affianca | 4000 | 24336 | 28336 |
| Sovrapponi | 4000 | 24337 | 28337 |
| Chiudi tutto | 4000 | 24338 | 28338 |
| Disponi icone | 4000 | 24335 | 28335 |
| cm_ActivateChild | 4000 | 24318 | 28318 |
| cm_ChildList | 4000 | 24319 | 28319 |
| Guida | 1000/3000 | 2/6 | 1002/3006 |
| Indice | 2000/4000 | 210 | 2210/4210 |
| Tastiera | 2000/4000 | 211 | 2211/4211 |
| Comandi | 2000/4000 | 212 | 2212/4212 |
| Procedure | 2000/4000 | 213 | 2213/4213 |
| Usare la guida | 2000/4000 | 214 | 2214/4214 |
| Informazioni | 2000/4000 | 220 | 2220/4220 |

*Figura 4 - Schema dei menu e delle relative opzioni, dei loro numeri identificativi e delle risultanti costanti di help, per la preparazione del file di risorse per MDIDEMO.*

scorso aggiungono al numero identificativo del menu o dell'opzione di menu su cui è posizionato l'utente: *ids_PopupMenu* (1000) se si tratta di un menu pop-up (un'opzione del menu principale) della *frame window*, *ids_MenuItem* (2000) se si tratta di un'opzione di un menu pop-up della *frame window*, *ids_ChildPopupMenu* (3000) e *ids_ChildMenuItem* (4000) per le corrispondenti situazioni delle *child window*.

Nella terza colonna trovate le costanti associate ad ogni menu e ad ogni opzione di questi. Per i menu non si tratta di altro che del loro numero d'ordine nella barra del menu principale; è questo il motivo per cui il menu Guida può valere sia 2 che 6: è infatti secondo nel menu della *frame window* ma sesto in quello delle *child*. Quanto alle opzioni, nel caso dei *system menu* si tratta di costanti predefinite (le trovate in WINDOWS.H), altrimenti sono valori definiti o da ObjectWindows (quelli da 24320 in su) o dal sottoscritto.

La quarta colonna, infine, ci dà i totali delle due precedenti: occorre definire una risorsa di tipo STRINGTABLE contenente coppie <numero, stringa> per ogni numero che compare nella quarta colonna; queste stringhe verranno poi visualizzate sulla riga di stato mediante la funzione API *LoadString*, chiamata da *TNewMDIFrame.WMEnterIdle*.

## L'Help Compiler del Turbo Vision

Avevo già in animo di parlare, tra qualche mese, della preparazione dei file di help, sia sotto Windows con la sua funzione *WinHelp*, sia con il TurboVision, con la unit HELPFILE e il programma TVHC presenti nella subdirectory TVDEMOS. È però successo che un abbonato a MC-link, Marco Cirinei di Marino, mentre preparava l'help per una sua applicazione in Turbo Vision, è incappato in un bug: un errore di esecuzione numero 204 (tipico di problemi con l'allocazione dinamica) proprio pochi giorni prima del termine per la consegna del suo lavoro. Ha esposto il problema nella rubrica PASCAL di MC-link, depositando anche il suo file di help in PAS-FILES, un'area collaterale alla rubrica e dedicata a messaggi lunghi o a file binari. Vista la sua urgenza, ho subito cercato un rimedio quale che fosse; grazie alle sue indicazioni sulle circostanze che procuravano l'errore, sono riuscito a suggerirgli in breve tempo una soluzione d'emergenza (in pratica: spezzare in due un lungo paragrafo), rinunciando ad ulteriori indagini.

Marco individuava poco dopo la ragione dei suoi guai e ne metteva a parte gli altri. A questo punto ho proposto a tutti una sfida: trovato il bug, vediamo chi trova la migliore soluzione. Il tutto comunicando malignamente che ne avevo già in mente almeno due... Quanto ne è seguito merita di essere riferito.

Vediamo prima la natura del bug. L'*Help Compiler* del Turbo Vision, TVHC, chiede in input un file ASCII in cui si succedano i vari testi, ognuno introdotto da una riga costituita dalla stringa «.topic» seguita dal nome simbolico con cui si designa ciò di cui il testo offre spiegazione (ad esempio: «.topic FileOpen»). Le righe del testo possono cominciare o no con uno spazio; se cominciano con uno spazio, verranno poi riprodotte nella finestra di help così come sono, altrimenti verranno composte in paragrafi le cui righe saranno automaticamente adattate alla larghezza della finestra di help (*word wrapping*). Ciò comporta che più righe consecutive che non inizino con uno spazio devono essere parcheggiate in un buffer per poter essere composte in un paragrafo, che verrà poi scritto nel file di help. Il problema è che il programma TVHC possiede un buffer di dimensione fissa (1024 byte), che può rivelarsi insufficiente nel caso di paragrafi di lunghezza superiore al previsto. Ecco anche perché la mia soluzione provvisoria funzionava: paragrafi più corti (i paragrafi, per essere distinti da TVHC, devono essere separati da una riga vuota) permettevano di evitare l'overflow del buffer.

Abbiamo scartato presto la soluzione più immediata: ricompilare TVHC dopo aver aumentato a «n» la dimensione del buffer. Sarebbe stato certo ragionevole ipotizzare che 4 o 6 KByte avrebbero evitato ogni problema, ma, ragionando in questo modo, non si sarebbe fatto altro che ripetere, su un piano diverso, lo stesso errore commesso dall'autore del programma. Sicuramente anche lui era convinto che i suoi 1024 byte erano una misura più che ragionevole.

Il problema, inoltre, presenta un interesse più generale: come scrivere un programma per il quale non si possa determinare a priori il volume massimo dei dati in input.

Accenno subito brevemente alla seconda delle soluzioni che avevo in mente: modificare la procedura *AddToBuffer*, in modo da provocare l'emissione di un messaggio d'errore nel caso di overflow, e rendere possibile determinare volta per volta la dimensione del buffer mediante un parametro «/b» della riga comando. Come dire: se mi arriva quel messaggio di errore, provo a ricompilare con un buffer più grande. Non particolarmente esaltante. Come se non bastasse, usavo la procedura *Val* per convertire da stringa a *word* il parametro, ma dimenticavo di verificare qui l'overflow (un 65536 diventava impunemente 1, e così via). L'errore mi è stato prontamente fatto notare da Salvatore Besso, di Reggio Emilia, che si è anche fatto carico di riscrivere una versione corretta del mio codice: si tratta di fare uso di una variabile temporanea di tipo *Longint*, come del resto raccomanda anche il manuale.

Ma in fondo si trattava solo della «seconda» soluzione.

## Array dinamici

L'altra soluzione era sicuramente più interessante: sostituire al buffer come array di byte, un buffer dinamico, capace di espandersi automaticamente secondo necessità. In sintesi, si tratta di cambiare il tipo della variabile *Buffer*, facendone un puntatore ad un buffer piuttosto che un array, e poi allocare un nuovo buffer più grande ogni volta che il vecchio risultasse insufficiente, copiare il contenuto del vecchio nel nuovo, assegnare a *Buffer* l'indirizzo del nuovo.

Il primo a proporla è stato Ottavio Risolia, di Osio Sotto, che però, per comprensibili problemi di tempo, si è limitato ad una esposizione in pseudocodice. Poco dopo è intervenuto anche Tommaso Masi, che ricorderete autore della serie di articoli dedicata allo SmallItalk. Tommaso ha preferito evitare la copia di un buffer vecchio in uno nuovo, sostituendo all'array di byte una lista di array: quando il primo è pieno, si alloca e si usa il secondo, e così via. Una soluzione elegante sia concettualmente che nell'implementazione, di cui l'autore ha praticamente fornito tutti i dettagli. Si potrebbe solo lamentare che né Ottavio né Tommaso hanno sottolineato la necessità di verificare che non venga superato il limite dei 65520 byte, limite che vale per ogni struttura allocata dinamicamente con un'unica istruzione *New* o *GetMem* (nel caso di Tommaso, la verifica andrebbe condotta sul campo *Text* della struttura *Paragraph*, definita in HELPFILE.PAS, in cui va riversato il contenuto del buffer; nel caso di Ottavio, ad ogni nuova allocazione di un buffer).

Ho provveduto io a fornire un'implementazione della soluzione delineata da Ottavio, che vi propongo nella figura 5. Vi ho aggiunto anche una precisazione di Marco (riga 1014) e la correzione di due altri piccoli bug che sono emersi durante i lavori: alla riga 253 va restituito il formato corretto alla stringa passata come secondo argomento alla procedura *FormatStr*, alla riga 998 non può essere chiamata la procedura *Error*, in quanto questa accede allo stream *HelpStrm* che viene però inizializzato solo alla riga 1012.

In sintesi: Ottavio ha proposto una soluzione che io ho implementato, Tommaso ha proposto una soluzione completa ma con una piccola omissione, Marco ha perfezionato la mia implementazione della prima soluzione, Salvatore ha corretto l'implementazione della mia seconda soluzione. Un vero e proprio lavoro di gruppo, che risulterà sicuramente utile a tutti quanti vorranno realizzare file di help per le loro applicazioni in Turbo Vision. Al punto che, terminati i lavori, non ho potuto che proclamare Marco Cirinei vincitore (puramente morale...) della sfida: il bug da lui individuato, infatti, ci ha permesso non solo di rendere più sicuro il funzionamento di TVHC, ma anche di trattare di array dinamici e... del corretto uso della procedura *Val*. MC

*Figura 5*
*Le correzioni da apportare al file TVHC.PAS.*

```
----- riga 253:
sostituire:  else FormatStr(S, '%s: %s %#3%s', L);

con:         else FormatStr(S, '%s: %s %3#%s', L);

----- righe 611-612: invece della costante, un nuovo tipo:
sostituire:  const
               BufferSize = 1024;

con:           PByteArray = ^TByteArray;
               TByteArray = array[0..65519] of Byte;

----- riga 614:
sostituire:  Buffer: array[0..BufferSize-1] of Byte;

con:         BufferSize: Word;
             Buffer: PByteArray;

----- procedura AddToBuffer, tutta nuova:
procedure AddToBuffer(var Line: String; Wrapping: Boolean);
var
  LL: Integer;
  Temp: PByteArray;
  OldSize: Word;
begin
  LL := Length(Line);
  if Ofs >= (BufferSize - LL) then begin
    OldSize := BufferSize;
    case BufferSize of
      65520: Error('Buffer Overflow');
      64512: BufferSize := 65520;
      else   Inc(BufferSize, 1024)
    end;
    GetMem(Temp, BufferSize);
    Move(Buffer^, Temp^, Ofs);
    FreeMem(Buffer, OldSize);
    Buffer := Temp;
  end;
  Move(Line[1], Buffer^[Ofs], LL);
  Inc(Ofs, LL);
  if Wrapping then Buffer^[Ofs] := Ord(' ')
  else Buffer^[Ofs] := 13;
  Inc(Ofs);
end;

----- riga 764:
sostituire:  Move(Buffer, P^.Text, Ofs);

con:         Move(Buffer^, P^.Text, Ofs);

----- riga 995, prima di { Calculate file names } aggiungere:
   BufferSize := 1024;
   GetMem(Buffer, BufferSize);

----- riga 998:
sostituire:  Error('File ' + TextName + ' not found.');

con       :  begin
               Writeln('File ' + TextName + ' not found.');
               Halt(1);
             end;

----- riga 1014:
prima di:    end.

inserire:    FreeMem(Buffer, BufferSize);
```

*Sergio Polini è raggiungibile tramite MC-link alla casella MC1166.*

# MICASOFT


**Via Pereira, 166 - 00136 Roma**

**Tel. (06) 3451443/3453382/3452048/348759 - FAX 3497295**

**MAGAZZINO CARICO E SCARICO MERCI A LARGO MACCAGNO, 27**


## OFFERTE SPECIALI

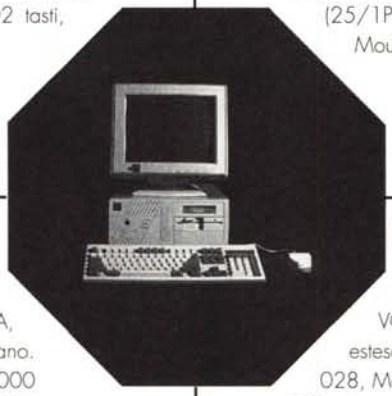
**286/40 VGA** Box Desk-Top, Scheda madre 286 16/21 Mhz, 1M RAM,RAM 80NS, Controller per 2HD+2FD, 1 Drive alta densità (3" 1/2 1.44), 1 Hard Disk 45 Mbyte 22 m.s., Scheda grafica VGA 56K, Scheda multifunzione (2S./1P.), Tastiera estesa 102 tasti, Mouse. DR DOS 5.0 Italiano.

Monitor Monocrom 0.31 L. 979.000 + IVA

Monitor Color 0.39 L.1.200.000 + IVA

Differenza per Scheda madre 20/26 Mhz L. 100.000

**386/SX** Box Desk-Top, Scheda madre 386 SX 25 Mhz, 1 Mega di RAM Controller per 2HD+2FD, 1 Drive alta densità (3" 1/2 1.44M) 1HD 45 Mbyte, 22 m.s., Scheda grafica VGA, 256K, Scheda Multifunzione (2S/1P.), Tastiera estesa 102 tasti, Monitor VGA colori 0.39, Mouse. DR DOS 5.0 Italiano. L. 1.480.000



**386/33 VGA** con 64K Cache memory 4Mb RAM, Controller per 2HD+2FD Hard Disk 80MB 15 m.s. FD 3" 1/2 1.4M, Scheda VGA 1Mbyte schede seriali, 1 parallela, tastiera estesa 102 tasti Monitor Super VGA, Multiscanner colore P 028, Mouse. DR DOS 5.0 Italiano. L. 2.250.000

Differenza per Scheda madre 386/40 Mhz L. 100.000

**486/33 VGA** con 256 Cache memory 4Mb RAM, Controller per 2HD+2FD Hard-Disk 125 Mbyte 15 m.s. FD 3" 1/2 1.4M, un FD 5" 1/4 1.2M, Scheda VGA 1Mbyte, 2 schede seriali, 1 parallela, Tastiera estesa 102 tasti, Monitor super VGA, Multiscanner colore P 028, Mouse. DR DOS 5.0 Italiano L. 2.290.000

Differenza per Hard disk 200 Mbyte 12 m.s. L. 360.000

Differenza per Hard disk 340Mbyte 10 m.s. L. 1.150.000

Differenza per Monitor 17" 1280x1024 0.26 L. 1.200.000

ECCO INOLTRE QUALCHE ARTICOLO ESTRATTO DAL NOSTRO LISTINO PREZZI:

**OFFERTA:**

**NOTE BOOK 386/33 Mhz, 64 K CACHE, 2 Mbyte HD 40 Mbyte, VGA**

**L. 4.400.000**

**DISTRIBUTORI PANASONIC**

**DISPONIBILE TUTTA LA GAMMA STAMPANTI E ACCESSORI**

**OFFERTA SPECIALE SUI MODELLI**

**KX-P1170 L. 350.000 KX-P1695 L. 799.000**

TUTTA LA GAMMA DEI PRODOTTI MICASOFT É GARANTITA 12 MESI

**MONITOR 1024x768**

| | |
|---|---|
| Monitor Monocrom VGA 14" F.B. pitch 031 | 175.000 |
| Monitor Color VGA Super 14" pitch 031 | 425.000 |
| Monitor Color VGA Super 14" pitch 028 | 485.000 |

**ACCESSORI**

| | |
|---|---|
| Floppy bulk 3" 1/2 2D | 595 |
| Floppy bulk HD 3" 1/2 | 1.100 |
| Floppy bulk 5" 1/4 HD | 900 |
| Scanner Genius | 200.000 |
| Porta Floppy a partire da | 6.000 |
| Cavi paralleli da 1.8 mt | 5.000 |
| Schermi antiriflesso a partire da | 16.500 |
| Mouse da | 19.000 |

**RICHIEDETE IL LISTINO VE LO SPEDIAMO GRATUITAMENTE.**

***Spedizioni in tutta Italia con un semplice ordine per telefono***

***Aperto dal lunedì al venerdì (9/13 - 14/18)***

**CERCHIAMO RIVENDITORI PER ZONE LIBERE - I PREZZI SI INTENDONO AL NETTO DI I.V.A. 19%**