ObjectWindows

di Sergio Polini (MC1166 su MC-link)

Windows 3.0 nel modo «386 avanzato», consente di aprire contemporaneamente diverse sessioni DOS, ognuna nella sua finestra. Analoga capacità troveremo in OS/2 2.0; è noto, d'altra parte, che quell'unica compatibility box delle versioni precedenti costituiva uno dei limiti di OS/2: nessuno si può permettere di ignorare l'enorme quantità di software nel mondo MS-DOS. Si stanno moltiplicando però anche le applicazioni per Windows 3.0; allo SMAU, ad esempio, ho visto diverse software house nostrane esporre le versioni per Windows dei loro prodotti per la contabilità, il computo metrico, ecc. Sta cioè prendendo forma un'altra realtà che nessuno potrà permettersi di ignorare. Ecco quindi che la Microsoft fa sapere che la API di Windows NT sarà al 95% la stessa della versione 3.0, che la IBM rilascerà prima o poi un OS/2 capace di eseguire direttamente le applicazioni Windows (senza bisogno di caricare prima Windows stesso). Se ne può concludere che chi sviluppi sotto DOS o Windows potrà comunque avere un mercato. Ancora meglio se sarà in grado di portare facilmente le proprie applicazioni dall'uno all'altro. La volta scorsa abbiamo iniziato a vedere come preparare dal lato DOS un tale porting, ora cominceremo ad occuparci del lato Windows

Programmare sotto Windows non è facilissimo. Da un certo punto di vista molti problemi vengono in verità drammaticamente semplificati: si dispone di una ricca serie di componenti già pronti per la costruzione dell'interfaccia utente, si possono quasi ignorare le differenti configurazioni hardware (monitor, scheda video, stampante, ecc.) su cui un programma dovrà girare. Si devono tuttavia fare i conti con una API *(Application Programming Interface)* decisamente complessa: più di 500 funzioni, più di 200 messaggi. A questo si aggiunge una fastidiosa incoerenza tra un'architettura concettualmente orientata all'oggetto e un'implementazione tradizionale.

ObjectWindows è la proposta Borland per limitare gli effetti di tale incoerenza: una gerarchia di classi che, tra l'altro, consente di costruire *oggetti d'interfaccia* che si sovrappongono agli *elementi d'interfaccia* offerti da Windows, per semplificarne la gestione. Va detto subito che la API di Windows non ne risulta tutta incapsulata, e questo rappresenta al tempo stesso un pregio e un limite di ObjectWindows; è un limite in quanto è praticamente impossibile scrivere un programma senza fare diretto ricorso a qualche funzione della API, è un pregio in quanto rimane comunque la possibilità di accedere a tutte le potenzialità di Windows. In concreto, usare ObjectWindows non vuol dire prescindere dalla API, con cui bisogna comunque fare i conti, ma lavorare in un ambiente in cui buona parte degli aspetti più abituali di un'applicazione vengono grandemente semplificati.

Due parole su MDI

Definita nelle specifiche SAA-CUA, l'interfaccia MDI prevede una finestra principale su cui si possono aprire più finestre secondarie, anche di tipo diverso. A differenza di quanto accade con altre finestre secondarie, quelle di un'applicazione MDI possono essere «massimizzate» o «minimizzate»; nel

```
program MDI;

{$R MDIAPP.RES}

uses WObjects, WinTypes, WinProcs;

type
  TMDIApp = object(TApplication)
    procedure InitMainWindow; virtual;
  end;

procedure TMDIApp.InitMainWindow;
begin
  MainWindow := New(PMDIWindow, Init('Applicazione MDI',
    LoadMenu(HInstance, 'MDIMenu')));
end;

var
  MDIApp: TMDIApp;

begin
  MDIApp.Init('MDIApp');
  MDIApp.Run;
  MDIApp.Done;
end.
```

Figura 1 - Lo schema di un'applicazione MDI realizzata con ObjectWindows.

primo caso occupano tutta l'area disponibile *(client area)* della finestra principale, nel secondo vengono ridotte a icona. Per inciso, la possibilità di operare in modo analogo sulle finestre di un'applicazione Turbo Vision mi pare confermi la sostanziale omogeneità tra questa e un'applicazione MDI, e quindi la validità del discorso che stiamo svolgendo. Ma andiamo avanti.

Il menu di un'applicazione MDI è al tempo stesso uno e multiplo: è unico in quanto (come anche in Turbo Vision) le singole finestre non possono avere una propria barra di menu, ma solo un titolo; è multiplo in quanto, potendo le finestre secondarie essere di tipo diverso, il menu della finestra principale non è necessariamente sempre lo stesso, ma può cambiare secondo il tipo della finestra secondaria di volta in volta resa attiva. È comunque tipico del menu di un'applicazione MDI un sottomenu intitolato *Windows* o *Finestre,* attraverso il quale operare sulle finestre secondarie accostandole l'una all'altra *(tiling),* sovrapponendole *(cascading),* chiudendole o selezionando ora l'una ora l'altra. Un'utile caratteristica di tale sottomenu è la sua dinamicità: ogni volta che si apre una nuova finestra, il suo titolo viene aggiunto alla lista delle opzioni, ed è proprio in questo modo che diventa possibile selezionare dal menu le diverse finestre. Si possono aggiungere fino a nove titoli; se si aprono più di nove finestre, viene aggiunta un'opzione attraverso la quale si può accedere ad una dialog box con una list box che consente di selezionare comunque la finestra desiderata.

Il Program Manager e il File Manager, Excel e WinWord, lo stesso Turbo Pascal per Windows, sono tutti esempi di applicazioni MDI. SysEdit, un programma nascosto nella subdirectory SYSTEM di Windows, utile per esaminare e modificare i file CONFIG.SYS, AUTOEXEC.BAT, WIN.INI e SYSTEM.INI, è un ulteriore esempio.

Lo schema di un'applicazione

Prima di vedere come lavorare su un'applicazione MDI, occorre mettere a fuoco alcuni concetti ed introdurre una

```
unit MDIUnit;

interface

uses WObjects, WinTypes, WinProcs;

const
  cw_MDIChildDestroy = wm_User;

type

  PMyMDIFrame = ^TMyMDIFrame;
  TMyMDIFrame = object(TMDIWindow)
    procedure InitClientWindow; virtual;
  end;

  PMyMDIClient = ^TMyMDIClient;
  TMyMDIClient = object(TMDIClient)
    procedure CWMDIChildDestroy(var Msg: TMessage);
      virtual wm_First + fw_MDIChildDestroy;
  end;

  PMyMDIChild = ^TMyMDIChild;
  TMyMDIChild = object(TWindow)
    constructor Init(AParent: PWindowsObject; ATitle, AMenu: PChar);
    procedure WMMDIActivate(var Msg: TMessage);
      virtual wm_First + wm_MDIActivate;
    procedure WMDestroy(var Msg: TMessage);
      virtual wm_First + wm_Destroy;
  end;

implementation

var
  HChildMenu: HMenu;

procedure TMyMDIFrame.InitClientWindow;
begin
  ClientWnd := New(PMyMDIClient, Init(@Self));
end;

procedure TMyMDIClient.CWMDIChildDestroy(var Msg: TMessage);
var
  HFrameMenu, HWinMenu: HMenu;
begin
  if LoWord(SendMEssage(HWindow, wm_MDIGetActive, 0, 0)) = 0 then begin
    HFRameMenu := PMDIWindow(Parent)^.Attr.Menu;
    HWinMenu := GetSubMenu(HFrameMenu, 0);
    SendMessage(HWindow, wm_MDISetMenu,
      0, MakeLong(HFrameMenu, HWinMenu));
    DrawMenuBar(Parent^.HWindow);
  end;
end;


constructor TMyMDIChild.Init(AParent: PWindowsObject;
                             ATitle, AMenu: PChar);
begin
  TWindow.Init(AParent, ATitle);
  if HChildMenu = 0 then
    HChildMenu := LoadMenu(HInstance, AMenu);
  Attr.Menu := HChildMenu;
end;

procedure TMyMDIChild.WMMDIActivate(var Msg: TMessage);
var
  HWinMenu: HMenu;
  i, n    : Word;
  Found   : Boolean;
begin
  if Msg.WParam <> 0 then begin
    n := GetMenuItemCount(Attr.Menu);
    Found := FALSE;
    while (n > 0) and (not Found) do begin
      HWinMenu := GetSubMenu(Attr.Menu, n - 1);
      if GetMenuState(HWinMenu, cm_TileChildren, mf_ByCommand) <> $FFFF
      then
        Found := TRUE;
      Dec(n);
    end;
    SendMessage(PMDIWindow(Parent)^.ClientWnd^.HWindow, wm_MDISetMenu,
      0, MakeLong(Attr.Menu, HWinMenu));
    DrawMenuBar(Parent^.HWindow);
  end;
end;

procedure TMyMDIChild.WMDestroy(var Msg: TMessage);
begin
  PostMessage(PMDIWindow(Parent)^.ClientWnd^.HWindow,
    cw_MDIChildDestroy, 0, 0);
  TWindow.WMDestroy(Msg);
end;

begin
  HChildMenu := 0;
end.
```

Figura 2 - Una unit che ridefinisce le classi TMDIWindow *e* TMDIClient, *e aggiunge una nuova classe per le* child window, *in modo da rendere possibile l'utilizzo di due diversi menu, secondo che sia aperta o no una* child window.

```
program MDI;

{$R MDI.RES}

uses WObjects, WinTypes, WinProcs, MDIUnit;

type

  TMDIApp = object(TApplication)
    procedure InitMainWindow; virtual;
  end;

  PFrameWindow = ^TFrameWindow;
  TFrameWindow = object(TMyMDIFrame)
    function InitChild: PWindowsObject; virtual;
  end;

procedure TMDIApp.InitMainWindow;
begin
  MainWindow := New(PFrameWindow, Init('Applicazione MDI',
    LoadMenu(HInstance, 'FrameMenu')));
end;

function TFrameWindow.InitChild: PWindowsObject;
begin
  InitChild := New(PMyMDIChild, Init(@Self,'Child Window','ChildMenu'));
end;

var
  MDIApp: TMDIApp;

begin
  MDIApp.Init('MDIApp');
  MDIApp.Run;
  MDIApp.Done;
end.
```

Figura 3 - Lo schema di un'applicazione MDI con un doppio menu.

specifica terminologia. Per prima cosa, osserviamo che le finestre secondarie non vengono aperte sulla finestra principale. Quest'ultima svolge il ruolo di mera cornice degli eventi (non a caso viene detta *frame window*) e la sua *client area* viene interamente coperta da un'altra finestra detta *client window*; le finestre secondarie, dette *child window*, possono essere aperte, posizionate, massimizzate e minimizzate solo nell'ambito della *client window* e attraverso essa, come avremo modo di vedere. Potrebbe sembrare che la *client window*, in quanto destinata a ricoprire la *client area* della finestra principale, sia solo un inutile doppione, ma così non è; è infatti possibile modificare le dimensioni della *client window* in modo da lasciare liberi dalle *child window* (anche quando massimizzate o iconizzate) spazi della *frame window*, che diventano in tal modo disponibili per altri scopi, quali una riga di stato o un *ribbon*. Le *child window*, infatti, «dipendono» dalla *client window*, mentre quanto viene aggiunto alla *frame window* «dipende» ovviamente da questa; ciò comporta che l'eventuale *ribbon* di un'applicazione (quale la striscia grigia di WinWord contenente pulsanti e combobox per i font, gli stili, ecc.) non viene coinvolto nelle operazioni di *tiling* o simili, governate dalla *client window*. Ne riparleremo.

Per ora consideriamo che, con Windows 3.0, realizzare un'applicazione MDI è relativamente semplice, grazie alle nuove potenti funzioni della sua API. Ciò nonostante, il sorgente dell'applicazione MDI «quasi minima» proposto da Petzold si sviluppa per ben tredici pagine: rimane comunque un gran lavoro da fare.

ui interviene ObjectWindows: una classe *TMDIWindow* per la *frame window* e una classe *TMDIClient* per la *client window* incapsulano tutta la funzionalità di un'applicazione MDI standard, come dimostra il programma MDIAPP.PAS fornito insieme al Turbo Pascal per Windows. Come potete verificare guardando la figura 1, si tratta di sole venticinque righe, di cui sei puramente ornamentali. L'enorme semplificazione dovrebbe risultare evidente, ma... non bisogna esagerare: per passare dallo schema della figura 1 ad una concreta applicazione c'è comunque da lavorare. MDIAPP, infatti, apre solo *child window* generiche; nel concreto, occorrerà derivare da *TWindow* le particolari finestre che vorremo utilizzare e ridefinire il metodo *InitChild* di *TMDIWindow*. E potrebbe non essere sufficiente.

Cambiare i menu

Consideriamo ad esempio il menu. In MDIAPP il menu è unico e non cambia secondo il tipo delle *child window;* questo non dipende dal fatto che le finestre di MDIAPP sono generiche istanze di *TWindow*, ma dall'assenza di un metodo che provveda ad installare ogni volta il menu corrispondente alla *child window* attiva. Questo è uno dei casi in cui occorre fare ricorso alla API, come ho fatto nella unit della figura 2. In particolare, occorre mandare alla *client window* il messaggio WM_MDISETMENU, sia quando viene attivata una *child window*, sia quando l'ultima *child window* viene chiusa.

Per semplificare il discorso, immaginiamo di volere un solo tipo di *child window*. Deriviamo quindi da *TWindow* una nuova classe, diciamo *TMyMDIChild*, il cui constructor accetti tre parametri: oltre agli usuali *AParent* e *ATitle*, anche il nome di un menu. Il menu viene caricato, e il suo handle assegnato prima alla variabile *HChildMenu* se questa vale ancora zero (come al momento della inizializzazione della unit), poi al campo *Menu* della variabile d'istanza *Attr*, ereditata da *TWindow*. In sostanza, il ruolo di *HChildMenu* è solo quello di evitare che uno stesso menu venga caricato più volte.

Per cogliere il momento in cui una *child window* viene attivata, occorre intercettare il messaggio WM_MDIACTIVATE; occorre quindi prevedere un metodo *WMMDIActivate*. Quel messaggio viene inviato da Windows alla *client window*, poi da questa sia alla *child window* che viene attivata (con *WParam* diverso da zero) che a quella che era prima attiva e viene ora disattivata (con *WParam* uguale a zero). Il metodo quindi non fa nulla se *Msg.WParam* è zero, ma, in caso contrario, percorre le opzioni del menu associato alla *child window* (il cui numero viene reso dalla funzione *GetMenuItemCount*) e i sottomenu corrispondenti ad ognuna di esse (ottenuti con *GetSubMenu*), per trovare quello in cui è presente un'opzione che provochi l'esecuzione del comando *cm_TileChildren*. Si tratta di un'opzione tipica del sottomenu *Finestre*, usata appunto per individuare questo. Ci si avvale della funzione GetMenuState che, dati un menu, l'identificativo di una sua opzione e un flag, ritorna $FFFF se quell'opzione non esiste nel menu (la docu-

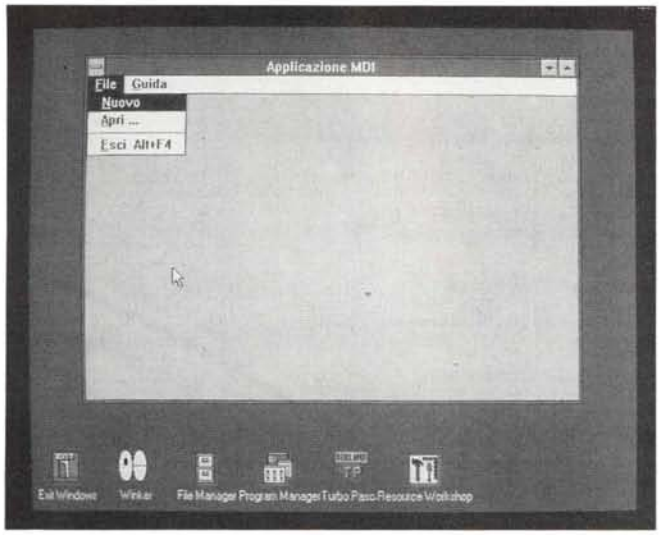

Figura 4 - Appena partito, il programma della figura 3 mostra un breve menu con due sole opzioni.

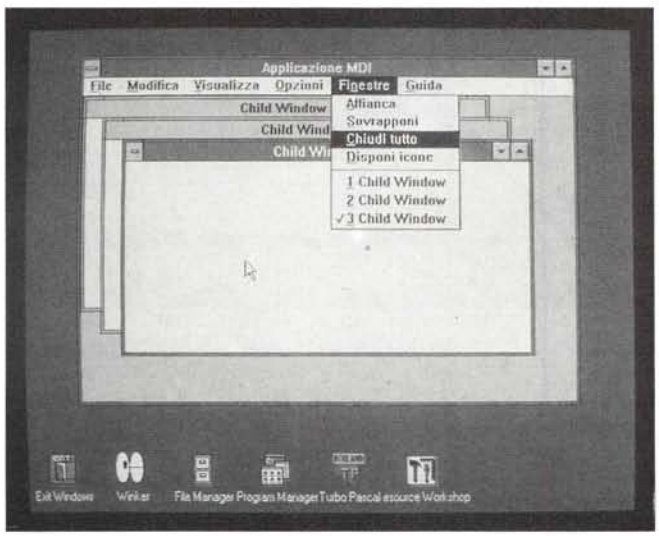

Figura 5 - Scegliendo l'opzione Nuovo *dal menu* File *del programma della figura 3, si possono aprire le* child window. *Ciò provoca la sostituzione del menu illustrato nella figura 4 con uno più ricco, comprendente tra l'altro una opzione* Finestre *per operare sulle* child window.

mentazione recita che, in tal caso, la funzione ritorna −1, ma, dato che il tipo del risultato della funzione è *Word*, si deve in realtà intendere $FFFF). Tutto ciò allo scopo di inviare alla *client window* il messaggio WM_MDISETMENU indicando sia il nuovo menu che il sottomenu di questo destinato ad allungarsi con i titoli delle finestre via via attivate.

Si deve però anche ripristinare il menu originario quando tutte le *child window* vengono chiuse. Per far questo, si può intercettare con un apposito metodo il messaggio WM_DESTROY che Windows manda ad una finestra prima di chiuderla; basta inviare alla *client window* un messaggio ad hoc e poi chiamare il metodo della classe base. Da notare che, per ottenere che il messaggio giunga a destinazione solo dopo che la *child window* viene chiusa, si usa la funzione *PostMessage* invece di *SendMessage*; questa infatti aspetta che il messaggio venga processato dal destinatario prima di restituire il controllo all'istruzione successiva, quella ritorna subito, limitandosi a porre il messaggio nella coda della finestra destinataria.

Ho usato il nome *cw_MDIChildDestroy* per il messaggio da mandare alla *client window*, non tanto perché in qualche modo «rende l'idea», ma perché è molto simile al nome coniato da Jeffrey Richter, dal cui libro sulla programmazione sotto Windows ho tratto utilissimi spunti (*Windows 3: A Developer's Guide*, M&T Books, 1991). Ho quindi derivato una classe da *TMDIClient* per aggiungere un metodo *CWMDIChildDestroy*. Qui si tratta in primo luogo di verificare se vi sono altre finestre aperte, o se quella che ha mandato il messaggio era l'ultima. Purtroppo non si può usare la variabile d'istanza *ChildList* in quanto, per motivi a me misteriosi, in ObjectWindows le finestre di volta in volta aperte vengono appese alla lista della *frame window* invece che della *client window*, nonostante dipendano da questa (il metodo *TileChildren* di *TMDIWindow*, ad esempio, non fa altro che chiamare l'omonimo metodo di *TMDIClient*). Non si può neppure usare la *ChildList* della *frame window*, in quanto potremmo decidere di aprire qualcos'altro in essa (ad esempio, una dialog box non modale per un ribbon). La *client window* manda quindi a se stessa il messaggio WM_MDIGETACTIVE, ottenendo da Windows un Longint in cui la word bassa contiene l'*handle* della finestra attiva o zero se non ve ne sono (la word alta è diversa da zero se la finestra è massimizzata). Se non vi sono più finestre aperte, si usa il messaggio WM_MDISETMENU, indicando come menu quello originario e il primo sottomenu di questo, quello con indice zero, come sottomenu *Finestre*.

Infine, occorre derivare una classe anche da *TMDIWindow*, per ridefinire il metodo *InitClientWindow* in modo che l'applicazione usi un'istanza della nuova classe.

Un nuovo schema

Ricorrere alla API di Windows non vuol dire rinunciare ai benefici della programmazione orientata all'oggetto. Una volta approntata la nostra unit, sarà facile servirsene per sviluppare applicazioni MDI con un doppio menu.

Occorre in primo luogo creare un file di risorse con due menu, chiamandoli ad esempio FrameMenu e ChildMenu. Si tratta poi di modificare lievemente lo schema della figura 1, aggiungendovi solo la definizione di una classe per la *frame window* tale che, nel suo metodo *InitChild*, chiami il constructor delle *child window* specificando il nome del secondo menu. In tutto trentasei righe, di cui nove vuote messe lì solo per non appesantire il listato.

Naturalmente ho semplificato le cose per brevità; sarebbe stato opportuno, ad esempio, prevedere una distinta unit per le *child window*, in modo da «nascondere» in essa quella variabile *HChildMenu* e rendere più chiara l'impostazione da dare ad un'applicazione con menu multipli. D'altra parte, non ci accontentiamo certo di quanto visto finora: da un lato dobbiamo vedere come gestire i menu multipli in un'applicazione Turbo Visione, dall'altro dobbiamo aggiungere almeno una riga di stato ad un'applicazione MDI.

MS

FLOPPERIA®

Viale Monte Nero, 15 • 20135 Milano
Telefono (02) 55.18.04.84 r.a.
Fax (02) 55.18.81.05 (24 ore)

Negozio aperto al pubblico dalle 10 alle 13 e dalle 15 alle 19. Vendita anche per corrispondenza.

Personal Computer EuroSys

Venduti in configurazione base (senza monitor), i nostri PC sono CONFIGURABILI SU MISURA, ovvero in base alle vostre preferenze, e sono coperti da una garanzia totale per 12 mesi. Scegliete VOI il tipo di monitor, a colori o monocromatico, la scheda grafica che preferite, la capacità dell'hard disk (tutti velocissimi, tempo d'accesso inferiore a 24 ms e trasferimento dati superiore ai 700 KB/sec.), la quantità di memoria Ram, e così via. Ecco due esempi di configurazione:

modello base 286-12	495.000	modello base 386-25	1.070.000
differenza scheda VGA	50.000	differenza scheda VGA	50.000
hard disk 20 MB	230.000	hard disk 45 MB	330.000
monitor VGA colori Philips	495.000	monitor colori 1024x768	595.000
Totale	1.270.000	Totale	2.045.000

Modelli base

I modelli base EuroSys sono composti dalla scheda CPU di vostra scelta e dalle seguenti parti: cabinet desk baby con alimentatore 200W, 1 disk drive 3"1/2 da 1.44 MB (5"1/4 da 360KB solo per modello 8088), 1 MB Ram (512 KB per modello 8088), scheda video duale (CGA più printer), tastiera USA estesa 101 tasti, controller per 2 floppy più 2 hard disk (solo 2 floppy per 8088).

8088 EuroSys 12 MHz 350.000
Norton SI: 4.6 • CPU Nec V20 compatibile 8088 • Espandibile a 640KB o 1 MB • Zoccolo per coprocessore opzionale 8087

286 EuroSys 12 MHz - Landmark: 16 MHz 495.000
286 EuroSys 20 MHz - Landmark: 26 MHz 595.000
CPU Intel 80286 16 bit • Espandibile a 4 MB, 0 wait states • Zoccolo per coprocessore opzionale 80287

386/20-SX EuroSys 20 MHz - Landmark: 26 MHz 880.000
386/25 EuroSys 25 MHz - Landmark: 33 MHz 1.070.000
386/33-C EuroSys 33 MHz cache 64K - Landmark: 56 MHz 1.360.000
CPU Intel 80386 32 bit • Espandibile a 8 MB, 0 wait states • Zoccolo per coprocessore opzionale 80387

486/20-SX EuroSys 20 MHz cache 128K - Landmark: 97 MHz 1.950.000
CPU Intel 80486 32 bit • Espandibile a 8 MB, 0 wait states • Zoccolo per coprocessore 487 opzionale

486/33 EuroSys 33 MHz cache 128K - Landmark: 167 MHz 2.490.000
486/33 EuroSys 33 MHz cache 128K EISA 3.930.000
CPU Intel 80486 32 bit • Espandibile a 8 MB, 0 wait states • Coprocessore 487 presente su scheda

Parti per configurazioni su misura

Configurazioni per 8088:	aggiungere
• scheda 2 seriali + 1 printer + 1 game	25.000
• cavo CGA-Scart per il collegamento ad un televisore Scart	70.000
• VGA 800x600 16 colori, 256 KB Ram video	50.000
• secondo disk drive 3"1/2 da 720K	75.000
• hard disk 20 MB e controller HD	400.000
• hard disk 40 MB e controller HD	690.000
• 640 KB Ram	30.000
• 1 MB Ram	50.000
Configurazioni per 286/386/486:	**aggiungere**
• secondo disk drive 5"1/4 da 1.2 MB	130.000
• scheda 2 seriali + 1 printer + 1 game	25.000
• mouse Genius GM-6	29.000
• mouse Agiler alta risoluzione 420-2100 dpi	75.000
• VGA 800x600 16 colori, 256 KB Ram video	50.000
• SuperVGA 1024x768 256 colori, 1 MB Ram video, chip Tseng ET-4000	200.000
• SuperVGA 1024x768 256 colori, 1 MB Ram video, chip Trident T-8900-C	150.000
• hard disk 20 MB Seagate 24 ms	230.000
• hard disk 45 MB Seagate 24 ms	330.000
• hard disk 85 MB Seagate 18 ms	545.000
• hard disk 130 MB Seagate 18 ms	770.000
• hard disk 210 MB Western Digital 14 ms	1.130.000
• hard disk 52 MB Quantum 17 ms	455.000
• hard disk 105 MB Quantum 17 ms	760.000
• hard disk 210 MB Quantum 15 ms	1.395.000
• hard disk 330 MB Seagate 15 ms	2.220.000
• controller IDE High Speed cache 4 MB Ram (0,5 ms)	1.060.000
• controller SCSI Future Domain	85.000
• hard disk SCSI 52 MB Quantum 17 ms	505.000
• hard disk SCSI 105 MB Quantum 17 ms	825.000
• hard disk SCSI 210 MB Quantum 15 ms	1.480.000
• hard disk SCSI 330 MB Maxtor 14 ms	2.330.000
• hard disk SCSI 670 MB Seagate 12 ms	3.160.000
• hard disk SCSI 1 GB Seagate 12 ms	4.640.000
• controller SCSI High Speed cache 4 MB Ram (0,4 ms)	1.660.000
• controller SCSI High Speed cache 4 MB Ram (0,4 ms) bus EISA	2.240.000
• opzione mirroring per controller High Speed SCSI	460.000
• cabinet deluxe e speed-display	50.000
• cabinet minitower 4 posizioni	120.000
• cabinet tower 5 posizioni	140.000
• tastiera italiana	25.000
per ogni 1 MB di Ram aggiuntivo	100.000

Hardware per PC

Motherboard XT 12 MHz	99.000
Motherboard AT 12 MHz	189.000
Motherboard AT 20 MHz	264.000
Motherboard 386/SX 20 MHz	484.000
Motherboard 386 25 MHz	730.000
Motherboard 386 33 MHz 64K cache	990.000
Motherboard 486/SX 20 MHz	1.560.000
Motherboard 486 33 MHz 128K cache	2.390.000
Motherboard 486 33 MHz EISA	3.920.000
SoundBlaster 2.0 — Nuova versione compatibile Windows 3.0 • Campiona a 44.1 KHz	299.000
SoundBlaster Pro 2.0 — Con CMS 12 voci stereo • MIDI Connector Box • Sequencer 64 tracce • Voice Editor • Bus a 16 bit	545.000
PC Scart — Cavo con interfaccia per collegare un PC con CGA ad un televisore con presa Scart	75.000
SuperVGA 1024x768 256 colori, 1 MB Ram video, chip Trident T-8900-C	199.000
SuperVGA 1024x768 256 colori, 1 MB Ram video, chip Tseng ET-4000	249.000
VGA 800x600 16 colori, 256 KB Ram video	99.000
CD Rom Chinon CDX-431 esterno	955.000
CD Rom Chinon CDS-431 interno	795.000
Coprocessore IIT 287 12 MHz	175.000
Coprocessore Intel 80287 XL 6-20 MHz	210.000
Coprocessore Intel 80387 25 MHz	490.000
Coprocessore Intel 80387 33 MHz	490.000
Coprocessore Intel 80387-SX 16 MHz	295.000
Coprocessore Intel 80387-SX 20 MHz	330.000
Coprocessore Intel 80487-SX 20 MHz	1.100.000
Kit HD 3"1/2 removibile per trasporto con borsa (standard IDE)	135.000
Kit HD 3"1/2 removibile per trasporto con borsa (standard SCSI)	162.000
Mouse Genius GM-6, seriale, optomeccanico	29.000
Mouse Agiler 200, risoluzione 250-2500 dpi, seriale	46.000
Mouse Agiler 530, ris. 420-2100 dpi, seriale, software Dr. Halo, anche PS/2	89.000
Mouse ottico Golden Image	96.000
Scanner A4, 400 dpi a toni di grigio, con software OCR	1.260.000
Tape streamer Teac 150 MB con controller e cavi (per MS-DOS, Xenix, Unix)	1.250.000
Tape streamer Teac 60 MB con controller e cavi (per MS-DOS, Xenix, Unix)	950.000

Monitor

Philips BM 7749 — Monitor Philips VGA mono 14" flat square	230.000
Philips CM 8833-II' — Monitor Philips colori 14", pitch 0.42 • 2 canali audio stereofonici • Ingressi CVBS, RGB lineare e TTL • Adatto per PC con CGA, Amiga e C-64	460.000
Philips CM 9032 VGA 640x480 — Monitor Philips colori 14" VGA, risoluzione 640x480, pitch 0.42 • Frequenza di scansione 31.5 KHz • Ingresso RGB con presa 15 pin • Completo di cavo	495.000
Philips CM 3209 VGA 1024x768 — Monitor Philips colori 14" VGA, risoluzione 1024x768, pitch 0.28, multiscan • Frequenze di scansione 31.5/35.2/35.5 KHz • Ingresso RGB con presa 15 pin • Completo di cavo	730.000
Philips CM 2789 MultiSync 20" 1280x1024 — Monitor Philips colori multisync 20", risoluzione 1280x1024 • Frequenze di scansione da 30 a 64 KHz	3.090.000
Crystal 14" Color MultiScan 1024x768	595.000
Sampo 19" Color MultiScan 1024x768 — Monitor a colori multiscan, pitch 0.28, risoluzione 1024x768 • Dotato di base basculante e completo di cavo	1.985.000
Crystal 14" Dual	190.000
Crystal 14" VGA 640x480	220.000
Crystal 14" MultiScan 1024x768 — Monitor monocromatico a fosfori bianchi • Schermo piatto • Disponibile in versione duale, VGA o multiscan	260.000

Modem Hayes compatibili

Esterno Discovery 300/1200/2400/Videotel	299.000
Esterno Discovery 300/1200/2400/Videotel + MNP5	435.000
Scheda Discovery 300/1200/2400/Videotel	255.000
Scheda Discovery 300/1200/2400/Videotel + MNP5	375.000
Modem Courier HST 19600 — Esterno con V42, V42bis e MNP5, per trasmissioni fino a 38400 Baud	1.475.000
Mo-Fax 96 portatile — Modem-fax esterno, si attacca ad un qualsiasi PC con porta seriale RS-232: fax send/receive 9600 Baud in multitasking, modem 300/1200/2400 Baud	550.000

Tutti i prezzi sono comprensivi di I.V.A.