I metodi di stampa della classe TPrinter

La volta scorsa abbiamo esaminato l'implementazione di un primo insieme dei metodi di TPrinter, *quelli che sovraintendono alla creazione e distruzione di un'istanza della classe, all'inizio e alla fine di una fase di stampa, all'impostazione della stampante, alla scelta del tipo carattere e dell'intervallo di tabulazione, all'inizio e alla fine di grassetto e sottolineato. Vedremo ora i metodi che vengono usati per stampare stringhe e caratteri e per posizionare la testina della stampante*

di Sergio Polini

Con ogni probabilità, *TPrinter.Print* è il metodo destinato ad essere usato con maggiore frequenza in ogni applicazione che usi la unit TVPRINT. Come si vede nella figura 1, *Print* non fa altro che chiamare il metodo *PrintChar* per ogni carattere della stringa che gli viene passata come argomento; *PrintChar* è uno dei due metodi nei quali più direttamente si affronta il problema della indipendenza dalle peculiarità di ogni stampante. L'altro è il metodo *Move*, usato sia direttamente, per muovere la testina di stampa fino ad una data posizione, sia per le tabulazioni.

Prima di inviare caratteri al dispositivo di output, si controlla sempre che siano verificate due condizioni: non deve essere intervenuto un errore di I/O e l'utente non deve aver interrotto la stampa premendo il tasto Esc. Questo vale anche per i metodi *PrintChar* e *Move*; l'uso del tasto Esc, inoltre, viene controllato proprio nel metodo *PrintChar*. Prima di procedere, quindi, sarà opportuno esaminare la gestione di errori e interruzioni.

Errori e interruzioni

Per scrivere sul dispositivo di output si usa la procedura *Write*, ogni volta preceduta da una direttiva $I– e seguita da una direttiva $I+; in questo modo ogni errore di I/O, compresi gli errori critici, anziché provocare l'arresto del programma assegna un codice d'errore ad una variabile interna, il cui valore può essere letto mediante la funzione *IOResult*.

Dopo ogni operazione di output, quindi, si verifica il suo esito mediante una funzione booleana *IOError* (figura 2),

```
procedure TPrinter.Print(S: String);
var
  i: Integer;
begin
  for i := 1 to Length(S) do
    PrintChar(S[i]);
end;

procedure TPrinter.PrintChar(Ch: Char);
var
  Event: TEvent;
begin
  if (not OkToPrint) or UserAbort then
    Exit;
  GetKeyEvent(Event);
  if (Event.What = evKeyDown) and (Event.KeyCode = kbEsc) then begin
    UserAbort := True;
    Exit;
  end;
  case Ch of
    chNewLine: NewLine;
    chNewPage: NewPage;
    chTab    : Tab;
    chCReturn: begin
                 (*$I-*) Write(Lst, Ch); (*$I+*)
                 PosX := 0;
                 OkToPrint := not IOError;
               end;
    else begin
      if (Ch < ' ') or (Ch = £127) or (PosX+CharWidth > MaxX) then
        Exit;
      (*$I-*) Write(Lst, Ch); (*$I+*)
      PosX := PosX + CharWidth;
      OkToPrint := not IOError;
      if OkToPrint and BFace and (Length(PD.BoldOn) = 0) then begin
        (*$I-*)
        Write(Lst, chBackSpace);
        Write(Lst, Ch);
        (*$I+*)
        OkToPrint := not IOError;
      end;
      if OkToPrint and ULine and (Length(PD.UlOn) = 0) then begin
        (*$I-*)
        Write(Lst, chBackSpace);
        Write(Lst, '_');
        (*$I+*)
        OkToPrint := not IOError;
      end;
    end;
  end;
end;
```

I metodi Print *e* PrintChar *della classe* TPrinter.

La funzione IOError.

```
function TPrinter.IOError: Boolean;
var
  Error: Word;
  Code : string[3];
begin
  Error := IOResult;
  if Error <> 0 then begin
    Str(Error, Code);
    MessageBox(£3'Problemi con la stampante'£13+
               £3'su '+PD.Setup.OutPort+£13+
               £3'Codice d''errore: '+Code,
               nil, mfOkButton+mfError);
  end;
  IOError := Error <> 0;
end;
```

che chiama *IOResult*. *IOError* ritorna FALSE se non si è verificato alcun errore, altrimenti, prima di ritornare TRUE, mostra un messaggio che rende noto all'utente il codice dell'errore. Il risultato di *IOError* viene sempre assegnato alla variabile *OkToPrint*, che può quindi essere usata per accertare la possibilità di proseguire la stampa.

Va notato che, in caso di errore critico (stampante spenta, off-line o senza carta, output su un file collocato su un dischetto protetto dalla scrittura, ecc.), scatta il *system error manager* di Turbo Vision, implementato nella funzione *SystemError*. Questa mostra un messaggio esplicativo dell'errore sulla riga di stato, proponendo di premere Invio e riprovare (dopo aver acceso la stampante, dopo aver tolto la protezione dal dischetto, ecc.), o Esc per rinunciare. Nel primo caso, se l'errore non si verifica più ne scompare ogni traccia: la funzione *IOResult*, infatti, ritorna zero ad indicare che tutto è andato per il meglio; nel secondo caso, invece, *IOResult* ritorna un valore diverso da zero.

La gestione degli errori critici viene inizializzata mediante la procedura *InitSysError*, chiamata automaticamente da *TApplication.Init*. È possibile controllare i colori della riga di stato con le variabili *SysColorAttr* e *SysMonoAttr*, o implementare un meccanismo diverso approntando una propria funzione e assegnandone l'indirizzo alla variabile *SysErrorFunc*, ma l'impianto di default mi è sembrato sostanzialmente adeguato.

Va costruito ex novo, invece, un meccanismo che consenta all'utente di interrompere una stampa.

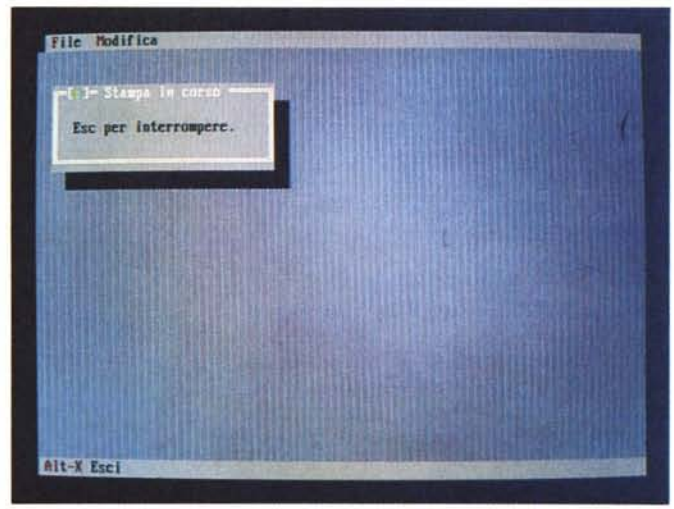
Ricorderete che il metodo *TPrinter Start*, dopo aver disabilitato comandi e

```
constructor TAbortDialog.Init(var Bounds: TRect; ATitle: TTitleStr);
var
  R: TRect;
begin
  TDialog.Init(Bounds, ATitle);
  R.Assign(3,2,26,3);
  Insert(New(PStaticText, Init(R, 'Esc per interrompere.')));
end;

function TAbortDialog.Valid(Command: Word): Boolean;
begin
  if Command = cmClose then begin
    UserAbort := True;
    DeskTop^.EnableCommands(SaveCommands);
    MenuBar^.SetState(sfDisabled, False);
  end;
  Valid := TDialog.Valid(Command);
end;
```

L'implementazione della classe TAbortDialog.

La dialog box per l'interruzione della stampa.



menu, apre una dialog box non modale istanza della classe *TAbortDialog*, che viene poi chiusa da *TPrinter.Finish*; il metodo *TAbortDialog.Valid*, chiamato automaticamente alla chiusura della dialog box, provvede a riabilitare comandi e menu (figura 3).

Mentre è visualizzata la dialog box (figura 4), l'utente non può eseguire alcun comando; se tuttavia preme il tasto Esc, si genera comunque un evento che viene intercettato dal metodo *PrintChar*. In questo, infatti, per prima cosa si controlla che *OkToPrint* sia vera e che *UserAbort* sia falsa; subito dopo si chiama *GetKeyEvent* per vedere se si è verificato un evento di tastiera; se si tratta della pressione del tasto Esc, si assegna TRUE a *UserAbort* e si esce.

Ciò comporta che, una volta che l'utente abbia premuto Esc, vengono ugualmente eseguite tutte le istruzioni di stampa, che però si fermano ai test su *OkToPrint* e *UserAbort*; se, ad esempio, si sta stampando un file, questo viene letto comunque tutto e si cerca di stampare tutte le sue righe. Per interrompere anche attività come la lettura di un file, ovviamente indipendenti dalla unit TVPRINT, si potrebbero rendere funzioni boolean le procedure *Print* e *PrintChar*, in modo da permettere l'uscita da un eventuale loop di stampa (come illustrato nella figura 5).

Stampa carattere per carattere

Una volta superati i test iniziali, il metodo *PrintChar* provvede a stampare il carattere passatogli come argomento, purché si tratti di un carattere stampabile (con codice ASCII uguale a 9, 10, 12 o 13, o compreso tra 32 e 255, 127 escluso) e così facendo non si superi il margine destro del foglio.

Alcuni caratteri vengono trattati in modo particolare: per *chNewLine* (ASCII 10), *chNewPage* (ASCII 12) e *chTab* (ASCII 9) si chiamano, rispettivamente, i metodi *NewLine*, *NewPage* e *Tab* (figura 6); per un *chCReturn* ASCII 13) si invia il carattere al dispositivo di output e si azzera la variabile *PosX* (posizione della testina di stampa sulla riga corrente). Quanto agli altri caratteri ammessi, si controlla prima che *PosX* non abbia raggiunto un valore tale che,

```
(* modifiche ai metodi Print e PrintChar nella unit TVPrint *)

function TPrinter.Print(S: String): Boolean;
var
  i: Integer;
  Result: Boolean;
begin
  i := 1;
  Result := TRUE;
  while (i <= Length(S)) and Result do begin
    Result := PrintChar(S[i]);
    Inc(i);
  end;
  Print := Result;
end;

function TPrinter.PrintChar(Ch: Char): Boolean;
var
  Event: TEvent;
begin
  if (not OkToPrint) or UserAbort then begin
    PrintChar := False;
    Exit;
  end;
  GetKeyEvent(Event);
  if (Event.What = evKeyDown) and (Event.KeyCode = kbEsc) then begin
    UserAbort := True;
    PrintChar := False;
    Exit;
  end;
  (* ... *)
  if (Ch < ' ') or (Ch = £127) or (PosX+CharWidth > MaxX) then begin
    PrintChar := False;
    Exit;
  end;
  (* ... *)
  PrintChar := OkToPrint;
end;

(* uso dei metodi in loop per la stampa di un file *)

begin
  (* ... *)
  Ok := True;
  while (not Eof(TextFile)) and Ok do begin
    Readln(TextFile, Line);
    Ok := Printer.Print(Line);
    Printer.NewLine;
  end;
  Printer.Finish;
  (* ... *)
end;
```

Esempio di una diversa implementazione dei metodi Print *e* PrintChar.

```
procedure TPrinter.NewLine;
begin
  PosX := 0.0;
  PosY := PosY + LineSpacing;
  if PosY > MaxY then
    NewPage
  else if OkToPrint and (not UserAbort) then begin
    (*$I-*) Writeln(Lst); (*$I+*)
    OkToPrint := not IOError;
  end;
end;

procedure TPrinter.NewPage;
begin
  if OkToPrint and (not UserAbort) then begin
    (*$I-*)
    Write(Lst, chCReturn);
    Write(Lst, chNewPage);
    (*$I+*)
    PosX := 0.0;
    PosY := 0.0;
    OkToPrint := not IOError;
  end;
end;

procedure TPrinter.Tab;
var
  NewPosX: Real;
begin
  NewPosX := Trunc(PosX / TabWidth) * TabWidth + TabWidth;
  if NewPosX <= MaxX then
    Move(NewPosX);
end;
```

I metodi NewLine, NewPage *e* Tab.

```
procedure TPrinter.Move(P: Real);
var
  SaveBFace, SaveULine: Boolean;
  i,N: Word;
  S: String[4];
begin
  if (P = PosX) or (P > MaxX) then
    Exit;
  if OkToPrint and (not UserAbort) then begin
    SaveBFace := BFace;
    SaveULine := ULine;
    if BFace then
      BoldFace(False);
    if not OkToPrint then Exit;
    if ULine then
      Underline(False);
    if not OkToPrint then Exit;
    if P < PosX then
      PrintChar(chCReturn);
    if not OkToPrint then Exit;
    P := P - PosX;
    while P >= CharWidth do begin
      PrintChar(' ');
      P := P - CharWidth;
    end;
    if not OkToPrint then Exit;
    if PD.Res > 0 then begin
      PosX := PosX + P;
      N := Round(P * PD.Res) + PD.Offset;
      if N > 0 then begin
        (*$I-*)
        if PD.Cmd[1] = '(' then
          for i := 1 to N do
            Write(Lst, PD.HeadOn)
        else begin
          Write(Lst, PD.HeadOn);
          case PD.Cmd[1] of
            'E': begin
                   Write(Lst, Chr(N mod 256));
                   Write(Lst, Chr(N div 256));
                   for i := 1 to N do
                     Write(Lst, Chr(0));
                 end;
            'A': begin
                   Str(N,S);
                   for i := 1 to Length(S) do
                     Write(Lst, S[i]);
                 end;
            'I': Write(Lst, Chr(N));
          end;
          Write(Lst, PD.HeadOff);
        end;
        (*$I+*)
        OkToPrint := not IOError;
      end;
      if PD.ResetFont = 0 then
        SetFont(Font);
    end;
        if SaveBFace then
          BoldFace(True);
        if SaveULine then
          Underline(True);
      end;
    end;
```

Il metodo Move.

aggiungendogli l'ampiezza del carattere (*CharWidth*), si andrebbe oltre il margine destro (*MaxX*); si stampa quindi il carattere e si aggiorna *PosX*.

Si provvede poi ad emulare il grassetto e il sottolineato nel caso l'uno o l'altro risultino attivati (variabili *BFace* o *ULine* con valore TRUE) e la definizione della stampante non comprenda le rispettive sequenze di controllo (*Length (PD.BoldOn)* o *Length(PD.UlOn)* uguali a zero): si porta indietro la testina di stampa inviando un *chBackSpace*, poi si ristampa il carattere per il grassetto, si invia un underscore (ASCII 95) per il sottolineato.

Va notato che, nell'implementazione del metodo, il gusto personale ha avuto influenza determinante. Vi sono, infatti, due soluzioni alternative per quanto riguarda il comportamento da assumere nel caso che la stringa da stampare termini oltre il margine destro: si può semplicemente troncare la stringa, non stampando affatto i caratteri «in eccesso», o si può intervenire con una chiamata di *NewLine* per proseguire sulla riga successiva. Ho preferito la prima soluzione, in quanto così non si rischia di sovvertire un'eventuale impaginazione e... perché mi piace di più. Nulla vieta, tuttavia, di optare per l'altra soluzione.

Posizionamento della testina di stampa

Il metodo *Move* (figura 7) consente di portare la testina di stampa in una qualsiasi posizione sulla riga corrente.

La posizione passata come parametro, che deve essere espressa in pollici, è da intendersi come posizione «assoluta»; con *Move(2.0)*, ad esempio, non si fa avanzare la testina verso destra di due pollici, ma la si porta a due pollici dal margine sinistro. Può quindi accadere che la nuova posizione sia a sinistra di quella corrente; in questo caso, si invia un *chCReturn*, che riporta al margine sinistro la testina e azzera la variabile *PosX*.

Dopo tali preliminari, la nuova posizione *P* è sicuramente a destra di quella corrente; viene quindi convertita in posizione «relativa», sottraendole *PosX*. Ciò consente di eseguire un loop controllato dalla condizione *P >= CharWidth*, in ogni ciclo del quale viene

Il dialogo di interruzione della stampa come proposto da Salvatore Besso.

```
constructor TAbortDialog.Init(var Bounds: TRect; ATitle: TTitleStr);
var
  R: TRect;
begin
  TDialog.Init(Bounds,ATitle);
  EventMask := EventMask or evBroadcast;
  R.Assign(1,2,43,7);
  Insert(New(PStaticText,Init(R,£3'Per interrompere la stampa:' +
                £13£13£3'Fare click con un pulsante del mouse' +
                £13£13£3'oppure premere un tasto qualsiasi')))
end;

procedure TAbortDialog.HandleEvent (var Event: TEvent);
var
  R: TRect;
  P: PPauseDialog;
begin
  TDialog.HandleEvent(Event);
  if (Event.What = evBroadcast) and (Event.Command = cmPause) then begin
    R.Assign(24,13,56,18);
    P := New(PPauseDialog,Init(R, 'Stampa in pausa'));
    SetState(sfActive, False);
    if Desktop^.ExecView(P) = cmAbort then
      UserAbort := True;
    SetState(sfActive, True);
    ClearEvent(Event)
  end
end;

constructor TPauseDialog.Init(var Bounds: TRect; ATitle: TTitleStr);
var
  R: TRect;
begin
  TDialog.Init(Bounds, ATitle);
  R.Assign(4,2,16,4);
  Insert(New(PButton,Init(R,'~C~ontinua',cmResume,bfDefault)));
  R.Assign(16,2,27,4);
  Insert(New(PButton,Init (R,'~T~ermina',cmAbort,bfNormal)));
  SelectNext(False)
end;

procedure TPauseDialog.HandleEvent (var Event: TEvent);
begin
  TDialog.HandleEvent (Event);
  if Event.What = evCommand then begin
    case Event.Command of
      cmResume: EndModal(cmResume);
      cmAbort: EndModal(cmAbort)
      else Exit
    end;
    ClearEvent(Event)
  end
end;
```

L'implementazione delle classi TAbortDialog *e* TPauseDialog *nella versione di Salvatore.*

stampato uno spazio e *P* viene decrementata sottraendole *CharWidth*. In questo modo, il movimento della testina si ottiene in buona parte con estrema semplicità ed è alla portata di qualsiasi stampante.

Rimane solo una frazione di movimento, per uno spostamento inferiore alla larghezza di un carattere, che richiede particolari attitudini dell'hardware. Prima di proseguire, quindi, si controlla che la stampante in uso sia capace di un movimento per punti, che abbia cioè una risoluzione *(PD.Res)* maggiore di zero.

Se così è, si calcola l'entità dello spostamento in punti, moltiplicando la sua ampiezza per la risoluzione, aggiungendo l'eventuale offset richiesto dalla sequenza di controllo della stampante e assegnando il risultato alla variabile *N*. Si procede quindi con il comando come scelto nella dialog box di definizione della stampante. Se la descrizione del comando *(PD.Cmd)* inizia con una parentesi tonda, il comando è «(n + offset) volte in comando Inizio», cioè *N* volte *PD.HeadOn*; negli altri casi va comunque inviato in primo luogo *PD.HeadOn*; quindi: se il comando inizia con una «E» si usa il metodo Epson, inviando un carattere con codice ASCII *N mod 256*, poi un carattere con codice *N div 256*, infine *N* caratteri con codice zero; se il comando inizia con una «A» si usa il metodo ANSI, inviando il numero *N* come stringa; se il comando inizia con «I» si intende «Inizio, CHR(n + offset)» e, quindi, si invia il carattere con codice ASCII pari a N.

Ricordo che lo spostamento è inferiore all'ampiezza di un carattere; anche ammettendo una risoluzione elevata come 600 punti per pollice e un tipo carattere ampio, con 4 caratteri per pollice, il numero dei punti non può superare 150 (un carattere occuperebbe un quarto di pollice, quindi 150 punti); ne segue che non è necessario verificare che *N* sia inferiore a 256. A meno che l'offset non sia «grande» (più di 100). Non ho ritenuto controllare il contemporaneo verificarsi delle tre condizioni (alta risoluzione, tipo di carattere ampio e offset grande), ma nulla vieta una maggiore prudenza.

Il contributo di Salvatore

Salvatore Besso è uno dei membri più attivi dell'area Pascal di MC-link. Ha potuto quindi esaminare da tempo la unit TVPRINT e ha proposto alcune modifiche, di cui vi propongo quella che mi è sembrata più interessante: un più flessibile dialogo di interruzione delle stampe.

Ha aggiunto tre costanti *cmPause*, *cmResume* e *cmAbort* per sospendere, riprendere o terminare una stampa, e ha modificato la prima parte del metodo *PrintChar*. Nella sua versione, una stampa può essere sospesa premendo qualsiasi tasto o premendo un qualsiasi pulsante del mouse (diventa così inutile la disabilitazione di menu e comandi in Start); in *PrintChar*, quindi, si chiamano prima *GetKeyEvent*, per verificare se *Event.What* è uguale a *evKeyDown*, poi *GetMouseEvent*, per verificare se *Event.What* è uguale a *evMouseDown*: in caso affermativo, prima si invia al *DeskTop* il comando *cmPause* con una istruzione *Message(DeskTop, evBroadcast, cmPause, nil)*, poi si esce se *UserAbort* vale TRUE.

Il messaggio viene intercettato dalla dialog box istanza di *TAbortDialog* (sempre aperta da Start), che reagisce aprendo una dialog box istanza di una classe *TPauseDialog* che propone la scelta tra proseguire o terminare la stampa. Vi propongo nella figura 8 l'aspetto delle due dialog box e nella figura 9 la loro implementazione come proposta da Salvatore (ho apportato solo modifiche di carattere estetico).

Potremmo terminare qui, in quanto l'implementazione dei restanti metodi è molto semplice. Ve la proporrò comunque il mese prossimo, a margine dell'introduzione al nostro nuovo argomento.

MC

Sergio Polini è raggiungibile tramite MC-link alla casella MC1166 e tramite Internet all'indirizzo MC1166@mclink.it.

PERSONAL SELF SERVICE
SUPERMARKET DELL'INFORMATICA

VENDITA - PERMUTE - NOLEGGIO PC
ASSEMBLATI NUOVI E USATI - SPEDIZIONI
POSTALI IN TUTTA ITALIA - ASSISTENZA TECNICA

WIN COMPUTER

SISTEMI INTEGRATIVI PER PERSONAL COMPUTER

CABINET E TASTIERE	ACCESSORI PER GRAFICA
MOTHER BOARD	MODEM E FAX
SCHEDE VIDEO	SISTEMI OPERATIVI
MONITOR	MEMORIE RAM
SCHEDE	STAMPANTI CITIZEN
HARD DISK IDE	HEWLETT PACKARD
FLOPPY DISK DRIVE	STAMPANTI OLIVETTI

PER I PRODOTTI SOPRA INDICATI
TELEFONARE PER QUOTAZIONI.
PREZZI DI IMPORTAZIONE!!

PRODOTTI MULTIMEDIALI CREATIVE LABS

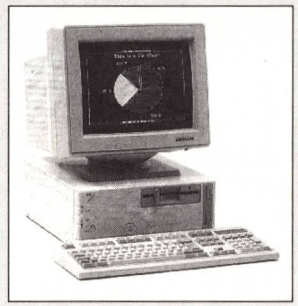
SOUNDMED ADLIB COMPAT.	95.000
SOUND BLASTER PRO BASIC	240.000
SOUND BLASTER 16 ASP	398.000
KIT INTERFACCIA MIDI	80.000
SCHEDA SCATTmini TELEV.	198.000
ADATTATORE VGA TO PAL	198.000
TITOLI SU CD ROM	65.000
COPPIA CASSE HI-FI SB	25.000
COPPIA CASSE HI-FI SB2	80.000
VIDEO BLASTER	545.000

EFFETTI VIDEO DIGITALI, WINDOWS 3.1 COMPATIBILE, AMPLIFICATORE, MICER STEREO, PCX, TIFF, BMP, ECC.

SOUND B. MULTIMEDIA KIT	940.000

BUSINESS KIT CONTIENE SCHEDA PRO, 2 C
LETTORE CD ROM INTERNO, 8 COMPACT DISK

PERSONAL COMPUTER PC WIN



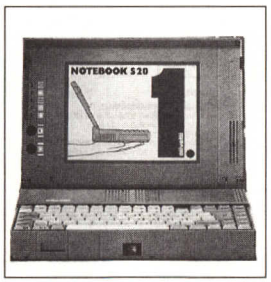
WIN COMPUTER 386SX/33
BOARD 386SX/33, 2MB RAM, CASE DESKTOP, FLOPPY DISK, DRIVE 1.44 MB, HARD DISK 40MB, MULTI I/O IDE 2 SER + 1 PAR., VGA 256 KB, MONITOR14", SVGA COLORI, TASTIERA, MOUSE.
LIRE 1.290.000

WIN COMPUTER 386SX/40
STESSA CONFIGURAZIONE
LIRE 1.330.000

WIN COMPUTER 386DX/40
BOARD 386DX/40, 2MB RAM, CASE DESKTOP, FLOPPY DISK DRIVE 1.44 MB, HARD DISK 107MB, MULTI I/O IDE 2 SER + 1 PAR., SCHEDA VGA 1 MB, MONITOR 14" SVGA COLORI, TASTIERA.
LIRE 1.695.000

WIN COMPUTER 486DX/33
256 CACHE, STESSA CONFIGURAZIONE
LIRE 2.280.000

SPECIALE NOTEBOOK

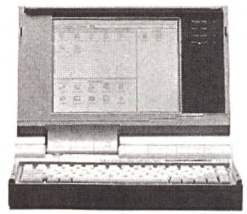


NOTEBOOK WIN 386SXL/20 S20
CPU 386SXL/20, 2 MB RAM, ESP. A 5, FDD 1,44 MB, HD 60 MB, VGA 32 T. G., MOUSE-PAD., RS232, BORSA, KG 2,9
LIRE 1.780.000

NOTEBOOK CHAPLET NBA386SX/25
CPU 386SX/25, 2 MB RAM, ESP. A 8 MB, FDD 1,44 MB, HD 80 MB, VGA 32 T.G., RSR 232, BATTERIA, PESO KG 2,9
LIRE 1.980.000

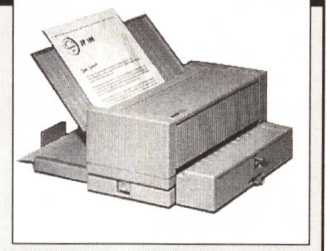
NOTEBOOK NP 204 486SLC/25-33
CPU 486SLC/25, 4 MB RAM FLOPPY 1,44, HD 120 MB, DISPLAY 10" VGA 64 T. GRIGIO, BORSA, KG. 2,9
LIRE 2.490.000

LAPTOP WIN 386DX/33 D33
CPU386DX/33 32K CACHE, 4 MB RAM FDD 1,44 MB, HDD 60 MB, VGA 12" 32 T. G., MOUSEPAD, 1 SLOT CARD
LIRE 2.190.000



Prodotti Olivetti

NOTEBOOK WIN 386SX/20
CPU 386SXL/ 20, 2 MB RAM, FLOPPY 1,44 MB, HD 60MB, VGA 32 T.GRIGIO, BORSA, KG 2,9
+
OLIVETTI BUBBLE INK-JET JP150
STAMPANTE A GETTO DI INCHIOSTRO 160 CPS
LIRE 2.090.000



VIA L. ZAMBARELLI, 16 - 00152 ROMA
TEL. (06) 58201066 - 58201067
FAX (06) 58201067
(ZONA MONTEVERDE)
FERMATA S. GIOVANNI DI DIO

VIA MATERA, 3 - 00182 ROMA
TEL. (06) 702.58.94/45.44/45.32
FAX (06) 757.39.21
(ZONA SAN GIOVANNI)
FERMATA METRO RE DI ROMA

ORARIO: 9.00/13.00 - 15.00/19.30 - SABATO MATTINA APERTO - I PREZZI SOPRA INDICATI SONO DA INTENDERSI IVA E MONTAGGIO ESCLUSI - IL PRESENTE LISTINO HA VALIDITÀ PER CAMBIO DOLLARO MASSIMO DI LIRE 1.400

OFFERTA DEL MESE

LETTORE CD ROM
SONY
LIRE 330.000

MONTAGGIO INTERNO COMPLETO DI SCHEDA DI CONTROLLO, CADDY, CAVI E DRIVE SOFTWARE