Non solo MS-DOS

di Sergio Polini

In passato ho spesso usato il corsivo iniziale per rispondere alle vostre lettere. Questa volta, tuttavia, mi sono giunte domande e osservazioni che non troverebbero adeguata risposta in poche righe; si tratta, inoltre, di questioni che mi sembrano di interesse generale. In questo e nel prossimo numero, quindi, non mi limiterò alla illustrazione della unit TVPRINT; tratteremo ora della importazione statica e dinamica di funzioni e procedure da una DLL con il Turbo Pascal per Windows, il mese prossimo vi riferirò dell'utile lavoro condotto da Fabrizio Superchi, di Brembate (Bergamo), circa alcuni bug contenuti nella unit EDITORS del Turbo Pascal per MS-DOS

Thomas Halva Labella, di Milano, che ringrazio per le belle parole con cui apre la sua lettera, mi racconta come l'articolo che ho dedicato all'uso delle stampanti sotto Windows (MC n. 117) gli abbia chiarito molti dubbi, ma gliene abbia procurato uno più generale: come si carica dinamicamente una procedura da una DLL? Sembrerebbe un problema banale, dal momento che il manuale propone esempi scarni ma completi di importazione sia statica che dinamica di funzioni e procedure da una DLL. Per una importazione statica, basta dichiarare la procedura (o funzione) come **external**, facendo seguire l'indicazione del nome della DLL e del nome o del numero ordinale della procedura; si può magari precisare che si deve operare diversamente secondo che la procedura venga dichiarata nella interfaccia di una unit o meno: nel primo caso, la sezione **interface** comprenderà la dichiarazione del nome e del tipo della procedura e dei suoi parametri senza la clausola **external**, che verrà specificata nella sezione **implementation** (come a pag. 130 della *Programmer's Guide*); negli altri casi, ad esempio quando la procedura viene dichiarata nel modulo **program**, occorre ricordare di utilizzare la clausola **far** prima di **external.**

Importazione dinamica da una DLL

Operando in questo modo, il nome di procedura utilizzato nel programma sarà sempre associato ad un'unica procedura di un'unica DLL; è però possibile (e a volte necessario, come appunto succede quando si usa una stampante) associare a quel nome una procedura tratta da una DLL che verrà determinata durante l'esecuzione (importazione dinamica). Si deve allora ricorrere alle funzioni *LoadLibrary* (seguita poi da *FreeLibrary*) e *GetProcAddress*, come illustrato a pag. 131 del manuale. Il problema in cui si è imbattuto Thomas sta proprio qui: l'esempio contenuto nel manuale non funziona.

Riporto nella figura 1 gli elementi essenziali dell'esempio. L'istruzione che assegna a *GetTime* l'indirizzo dell'omonima procedura della DLL non viene compilata, ma viene segnalato un errore di *invalid type cast* (ricordo che il tipo del risultato di *GetProcAddress* è *TFarProc*). Trattandosi di un «indirizzo di procedura» che viene assegnato ad una varabile che non è un puntatore, verrebbe spontaneo dereferenziare il risultato della chiamata di *GetProcAddress*, come indicato nella figura 2, e si riuscirebbe a compilare (!), per ottenere poi un errore durante l'esecuzione.

In realtà, nonostante il Turbo Pascal supporti pienamente il *typecasting* anche con i tipi procedurali, nonostante consenta quasi sempre di utilizzare direttamente il risultato di una funzione, non si deve dimenticare che ciò che si sottopone a *typecasting* deve essere una variabile a tutti gli effetti, ovvero un oggetto con un proprio indirizzo in me-

```
type
  TTimeRec = record ... end;
  TGetTime = procedure(var Time: TTimeRec);
var
  Time: TTimeRec;
  Handle: THandle;
  GetTime: TGetTime;
begin
  Handle := LoadLibrary('DATETIME.DLL');
  ...
     GetTime := TGetTime(GetProcAddress(Handle, 'GETTIME'));
     ...
        GetTime(Time);
  ...
end.
```

Figura 1 - Gli elementi essenziali dell'esempio proposto dalla Programmer's Guide *del Turbo Pascal per Windows circa l'importazione dinamica di funzioni da una DLL. L'esempio però non viene compilato.*

moria, mentre il risultato di una funzione viene spesso parcheggiato nei registri del microprocessore. Per meglio comprendere e ricordare tale limitazione, si può pensare a quello che succede quando una funzione o procedura richiede un parametro variabile: le si può passare una variabile, o anche il risultato dereferenziato di una funzione che ritorni un puntatore, ma non direttamente il risultato di una funzione (il programmino della figura 3 ce lo conferma). Analogamente, si può operare un *typecasting* su una variabile o sul risultato dereferenziato di una funzione che ritorni un puntatore, ma non direttamente sul risultato di una funzione; non si può quindi convertire direttamente in tipo *TGetTime* il risultato di *GetProcAddress*.

Vi sono diverse soluzioni. La più immediata consiste nell'utilizzo di una variabile di tipo *TFarProc* cui assegnare il risultato di *GetProcAddress*, per poi potere appunto operare il *typecasting* su una variabile (figura 4). Si può anche ricorrere ad una soluzione forse più sintetica, ma di chiarezza non esemplare, ovvero sostituire l'istruzione che non compila con quella proposta nella figura 5 (suggerita a pag. 55 della *Programmer's Guide*).

```
Handle := LoadLibrary('DATETIME.DLL');
...
   GetTime := TGetTime(GetProcAddress(Handle, 'GETTIME')^); (* <-- *)
   ...
      GetTime(Time);
```

Figura 2 - Un primo tentativo di soluzione. Il programma viene compilato, ma provoca un errore durante l'esecuzione.

Turbo Debugger e Windows 3.1

Nella sua lettera, Thomas propone altre osservazioni interessanti. Racconta ad esempio che, dopo aver letto tutta la manualistica, è rimasto con due punti oscuri: come leggere, visualizzare e scrivere file BMP e come utilizzare una stampante. Abbiamo già visto qualcosa circa quest'ultimo argomento; quanto ai file BMP, per ora gli consiglio di studiarsi i sorgenti di alcuni demo forniti col compilatore, come STRETCH.PAS. Non è escluso, comunque, che si torni su entrambi gli argomenti nell'ambito della rubrica.

Thomas riferisce anche dell'impossibilità di usare il Turbo Debugger da quando utilizza Windows 3.1. In realtà la nuova versione di tale ambiente pone anche altri problemi, di minore portata.

Purtroppo, contrariamente a quanto normalmente è lecito attendersi, non sempre noi utenti ricaviamo benefici dalla concorrenza tra i produttori. Windows 3.1 è forse un esempio di tali eccezioni: nato in un contesto di vivacissima polemica tra Microsoft e IBM, porta probabilmente in sé le conseguenze di una politica che ha sacrificato l'esigenza di compatibilità con le applicazioni già esistenti a quella di non essere... troppo compatibile con OS/2. Ne è risultato, ad esempio, non solo che il Turbo Debugger non funziona più, ma anche l'occasionale imperfetto aggiornamento della riga di stato del Turbo Pascal.

Le soluzioni sono molteplici. In primo luogo, la Borland ha rilasciato una versione 1.5 del compilatore, nella quale non solo vengono risolti tali problemi, ma si offrono anche un editor migliorato con *syntax highlighting*, il supporto alle nuove potenzialità di Windows 3.1 (dall'OLE ai font *True Type*) e il potente Resource Workshop in sostituzione del Whitewater Resource Toolkit. Chiunque possieda il TPW 1.0, può ricevere la nuova versione sopportando un contenuto costo di *upgrade*. La documentazione cartacea lascia in verità a desiderare, in quanto è praticamente identica a quella della versione precedente; in particolare, solo nell'help in linea si possono trovare notizie sulla nuova API; a parziale compensazione, la Borland propone la documentazione completa dell'API di Windows 3.1, in inglese o in italiano, con esempi sia in C/C++ che in Pascal, ad un prezzo sicuramente interessante.

Chi non volesse acquistare il Turbo Pascal per Windows 1.5, può prelevare dall'area BORLAND-FILES di MC-Link il file TPWIN31.ZIP, che contiene un programma di patch per l'IDE, le unit d'interfaccia alla nuova API, alcuni demo per OLE e font *True Type*, la nuova versione del Turbo Debugger per Windows e di WINDEBUG.DLL. Su MC-link, nella stessa area, si trova anche TP602:ZIP, un file contenente un aggiornamento di tutte le unit del Turbo Pascal per DOS (sorgenti compresi), mediante il quale si trasforma il compilatore nella versione 6.02; non vi sono differenze sostanziali rispetto alla versione 6.0, tranne la correzione di alcuni bug (ma non di quelli che vedremo nel prossimo numero...).

Infine, si potrebbe anche decidere di girare pagina. La Borland ha infatti recentemente presentato un prodotto completamente rifatto, il Borland Pascal 7.0. Si tratta di una confezione comprendente nuove versioni sia del compilatore per MS-DOS che di quello per Windows, entrambe dalle caratteristiche molto interessanti. Al momento in cui scrivo, non sono in grado di dirvi se

```
Program VarParam;
type
  PInteger = ^Integer;
var
  i: integer;
function Pippo: Integer;
  begin
  end;
function Pluto: PInteger;
  begin
  end;
procedure Paperino(var n: Integer);
  begin
  end;
begin
  i := 3;
  Paperino(i);       (* corretto *)
  Paperino(Pippo);   (* sbagliato *)
  Paperino(Pluto^); (* corretto *)
end.
```

Figura 3 - Un esempio che ci aiuta a capire la natura del problema. Non si può passare il risultato di una funzione ad una funzione o procedura che vuole un parametro variabile. Analoghe restrizioni valgono per il typecasting.

Figura 4 - Una prima soluzione: assegnare il risultato della funzione ad una variabile ausiliaria e operare poi il typecasting *su questa.*

```
var
  FP: TFarProc;
...
   FP := GetProcAddress(Handle, 'GETTIME');
   (* e poi: *)
   GetTime := TGetTime(FP);
   ...
      GetTime(Time);
   (* oppure, facendo a meno della variabile GetTime: *)
      TGetTime(FP)(Time);
```

```
@GetTime := GetProcAddress(Handle, 'GETTIME');
if @GetTime <> nil then begin
  GetTime(Time);
  ...
```

Figura 5 - Una soluzione alternativa, ma forse poco soddisfacente quanto a chiarezza, viene suggerita a pag. 55 della Programmer's Guide.

potete trovare più precise notizie al riguardo in questo stesso numero; in caso negativo, consideratevi convocati per gennaio!

La unit TVPRINT

Torniamo ora alle nostre stampanti sotto MS-DOS. La figura 6 riproduce l'interfaccia della unit TVPRINT, mediante la quale è possibile pilotare qualsiasi stampante installata con il programma PRINST, prescindendo dalle sue particolarità.

Devo ripetere che la unit non rappresenta una soluzione definitiva del problema. Potrei ricordare, in primo luogo, quanto osservato a suo tempo da Brad J. Cox circa la «maturità» di una classe: nella prima fase del suo disegno, una classe va considerata come una prima approssimazione, in quanto solo il suo uso effettivo in numerose applicazioni, solo la concreta verifica della sua capacità di prestarsi alla derivazione di sottoclassi, possono portare ad una stabile definizione della sua interfaccia e della sua implementazione. A ciò devo aggiungere che, per i motivi di brevità più volte esposti, ho limitato la classe *TPrinter* all'essenziale, riservandomi di suggerire sinteticamente alcune possibili estensioni. Tanto per dare subito un esempio, richiamo la vostra attenzione su due variabili d'istanza, *CharWidth* e *LineSpacing*; alla prima viene di volta in volta assegnata la larghezza dei caratteri nel font corrente, ma, se si volessero gestire anche font proporzionali, potrebbe ben preferirsi una funzione che leggesse da una tabella l'ampiezza di ogni singolo carattere; la seconda, come vedremo, viene in pratica trattata come una costante, in quanto le viene assegnato un valore di interlinea corrispondente a sei righe per pollice e tale valore non viene mai modificato; non sarebbe difficile, tuttavia, prevedere valori diversi e variabili se solo si aggiungesse in PRDEF.PAS la definizione delle sequenze di controllo da inviare alla stampante per variare l'interlinea.

L'interfaccia della unit comprende unicamente la dichiarazione di alcune costanti (i codici dei caratteri per il *backspace*, la tabulazione e gli avanzamenti di riga e di pagina) e della classe *TPrinter*. In questa troviamo una variabile d'istanza pubblica *PD* di tipo *TPrinterDef*, seguita da metodi raccolti in diversi gruppi. Il constructor e il destructor si incaricano della creazione e distruzione di un'istanza della classe, mentre i metodi *Start*, *Reset* e *Finish* provvedono all'inizio e al termine di una singola fase di stampa. Seguono metodi per cambiare l'impostazione di quelle che abbiamo definito caratteristiche temporanee della stampante (ad esempio, il dispositivo di output), per scegliere il font e l'intervallo di tabulazione, per attivare o disattivare grassetto o sottolineato, per la stampa di stringhe e caratteri, per avanzare di una riga o di una pagina, per il

```
unit TVPrint;
(*$X+*)
interface

uses Objects, Drivers, Views, Dialogs, PrSetup;

const

  chBackSpace = #08;
  chTab       = #09;
  chNewLine   = #10;
  chNewPage   = #12;

type

  PPrinter = ^TPrinter;
  TPrinter = object(TObject)
    PD: TPrinterDef;                    (* definizione della stampante *)

    constructor Init;                   (* inizio/fine istanza *)
    destructor Done; virtual;

    procedure Start;                    (* inizio/fine stampa *)
    procedure Reset;
    procedure Finish;

    procedure DoSetup;                  (* impostazione stampante *)

    procedure SelectFont;               (* scelta font, intervallo *)
    procedure SetFont(F: Word);         (* di tabulazione, grassetto *)
    procedure SetTab(T: Real);          (* e sottolineato *)
    procedure BoldFace(On: Boolean);
    procedure Underline(On: Boolean);

    procedure Print(S: String);         (* stampa *)
    procedure PrintChar(Ch: Char);
    procedure NewLine;
    procedure NewPage;

    procedure Move(P: Real);            (* posizione testina di stampa *)
    procedure Tab;

    function IOError: Boolean;          (* gestione errori *)

    function GetMaxX: Real;             (* funzioni di accesso *)
    function GetMaxY: Real;
    function GetPosX: Real;
    function GetPosY: Real;
    function GetTab: Real;
    function GetFont(F: Word): PFont;
    function GetFontCount: Word;
    function TextLength(S: String): Real;
private
  Lst: Text;                            (* porta/file di output *)
  MaxX, MaxY: Real;                     (* dimensioni area di stampa *)
  PosX, PosY: Real;                     (* coordinate testina di stampa *)
  TabWidth: Real;                       (* intervallo di tabulazione *)
  Font: Integer;                        (* numero d'ordine font corrente *)
  CharWidth: Real;                      (* ampiezza carattere *)
  LineSpacing: Real;                    (* interlinea *)
  OkToPrint: Boolean;                   (* falso se IOResult <> 0 *)
  BFace, ULine: Boolean;                (* flag grassetto/sottolineato *)
end;
```

Figura 6 - L'interfaccia della unit TVPRINT.

posizionamento della testina di stampa. La sezione pubblica si chiude con funzioni per la gestione degli errori e per l'accesso alle variabili private.

La figura 7 riproduce le prima parte della sezione **implementation** della unit; in essa troviamo la dichiarazione di alcune costanti e variabili, nonché di altre classi utilizzate da *TPrinter*.

La classe *TPrSelectDialog* viene usata dal metodo *TPrinter.DoSetup* per proporre all'utente la scelta tra una delle stampanti installate e per consentirgli di cambiarne l'impostazione, mediante una dialog box come quella che abbiamo visto il mese scorso. L'elenco delle stampanti installate, mostrato in una list box, viene realizzato mediante una collezione di stringhe; poiché si tratta di stringhe «normali» (non di istanze di una classe derivata da *TObject*), è necessario ridefinire il metodo *FreeItem* della classe *TCollection* e, quindi, approntare una classe derivata da questa, *TPrinterCollection*. Si sarebbe anche potuto utilizzare la classe *TStringCollection;* una questione di gusti.

La classe *TFontSelectDialog* viene usata dal metodo *TPrinter.SelectFont* per proporre all'utente la scelta tra uno dei tipi carattere disponibili sulla stampante correntemente selezionata, mediante una dialog box anch'essa mostrata il mese scorso.

La classe *TPrintFileDialog* viene usata dal metodo *TPrinter.Start* nel caso l'utente abbia configurato la stampante per un output su file; viene aperta una dialog box con la quale si chiede il nome del file; se il file già esiste, viene chiesta conferma.

La classe *TAbortDialog,* infine, definisce una dialog box non modale che viene aperta da *TPrinter.Start* e poi chiusa da *TPrinter.Finish*; mentre rimane aperta, consente all'utente di interrompere la stampa premendo il tasto Esc, che viene intercettato da *TPrinter.PrintChar* con conseguente assegnazione di TRUE alla variabile *UserAbort*. La dialog box non è modale per consentire la prosecuzione delle operazioni di stampa; perché solo tali operazioni siano consentite, *TPrinter.Start* salva temporaneamente nella variabile *SaveCommands* e poi disabilita tutti i comandi, che vengono ripristinati quando la dialog box viene chiusa.

Il mese prossimo inizieremo l'esame dell'implementazione di tali classi. MC

```
implementation

uses Dos, App, MsgBox;

const
  chCReturn = #13;
  cmSetup   = 254;

type
  PPrinterCollection = ^TPrinterCollection;
  TPrinterCollection = object(TCollection)
    procedure FreeItem(Item: Pointer); virtual;
  end;

  PPrSelectDialog = ^TPrSelectDialog;
  TPrSelectDialog = object(TDialog)
    PP: PPrinter;
    LB: PListBox;
    PR: TPrinterDef;
    PrList: PPrinterCollection;
    constructor Init(var Bounds: TRect; Prn: PPrinter);
    destructor Done; virtual;
    procedure HandleEvent(var Event: TEvent); virtual;
  end;

  PFontSelectDialog = ^TFontSelectDialog;
  TFontSelectDialog = object(TDialog)
    PP: PPrinter;
    FB: PFontBox;
    constructor Init(var Bounds: TRect; AFont: Integer; Prn: PPrinter);
    procedure HandleEvent(var Event: TEvent); virtual;
  end;

  PPrintFileDialog = ^TPrintFileDialog;
  TPrintFileDialog = object(TDialog)
    FName: PInputLine;
    constructor Init(var Bounds: TRect);
    function Valid(Command: Word): Boolean; virtual;
  end;

  PAbortDialog = ^TAbortDialog;
  TAbortDialog = object(TDialog)
    constructor Init(var Bounds: TRect; ATitle: TTitleStr);
    function Valid(Command: Word): Boolean; virtual;
  end;

var
  SaveCommands: TCommandSet;
  AbortDialog : PAbortDialog;
  UserAbort   : Boolean;
```

*Figura 7 - Costanti, tipi e variabili dichiarati nella sezione **implementation** della unit TVPRINT.*

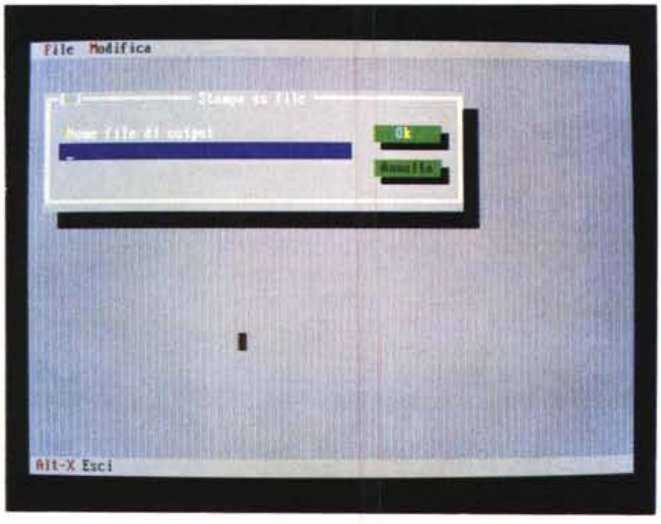

Figura 8 - La dialog box con la quale si chiede di indicare il file di output, nel caso la stampante sia configurata per un output su file.

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