Turbo Pascal per Windows

di Sergio Polini

Fantastico. Prassi vorrebbe che il giudizio sul prodotto venisse espresso nelle conclusioni, dopo averne esaminato criticamente le diverse caratteristiche. Ma qui non si riesce proprio a trattenere l'ammirazione. Non occorre una prova pesante e minuziosa: appena completata l'installazione (facile e veloce), non si può resistere alla tentazione di provare qualcosa. In fondo, per quanto i manuali siano ovviamente indispensabili, basta avere un minimo di esperienza con le versioni per DOS del Turbo Pascal e con Windows 3.0 per essere in grado di provare qualcosa. Con pochi colpi di mouse si apre la dialog box di «File/Open», ci si porta su una directory come «docdemos», si sceglie un file a caso. L'interfaccia MDI fornita da Windows ci propone subito una finestra di editing aperta su quel file. Sempre con il mouse si sceglie «Run/Run», o magari si preme l'abituale combinazione Ctrl-F9: in un attimo il programma viene compilato e si apre la sua finestra per dare inizio all'esecuzione. E non ho parlato a caso: ho detto «in un attimo» perché per tutto questo ci vuole davvero meno a farlo che a descriverlo. Mi tornano in mente le parole di un recensore americano alle prese con una versione ormai storica del Turbo Pascal: il primo compilatore integrato veniva messo a confronto con il linguaggio allora più diffuso, con l'interprete Basic fornito insieme con il DOS; un linguaggio il cui pregio fondamentale era forse dato dalla immediatezza con cui, digitando «RUN», si poteva verificare se il proprio programma girava a dovere. Il Turbo Pascal, notava quel recensore, nonostante fosse un compilatore e non un interprete, era tanto veloce che a compilare ci voleva meno che a digitare «RUN». Anders Hejlsberg, creatore di quel Turbo Pascal, nonché tuttora principale ispiratore delle strategie Borland per i linguaggi, ha ripetuto il miracolo.

Magari penserete che sto esagerando. Ma voi avete mai provato a pro-

grammare sotto Windows? Si tratta di un ambiente ideale per molteplici aspetti: indipendenza dall'hardware (mai più quei programmi ognuno con i propri driver per le schede video e le stampanti!), standardizzazione dell'interfaccia utente, multitasking, scambio di dati tra diverse applicazioni, mega e mega di memoria potenzialmente a disposizione (altro che limite di 640K!) e, in difetto di RAM, overlay non più necessario in quanto gestito in modo trasparente dallo stesso Windows, ecc. Ma programmare in tale ambiente è stato finora un processo lungo e penoso: file di «definizioni» e di «risorse» accanto ai sorgenti, pesanti convenzioni da seguire anche per il più banale programmino (mai meno di cento righe!), un ciclo di compilazione appesantito dalla necessità di usare sempre il Resource Compiler *e di lavorare sotto DOS, la necessità di un secondo monitor per il debugging simbolico, ecc. Ecco perché il primo maggio a San Francisco, quando Anders ha illustrato il Turbo Pascal per Windows nella sala Ralston dell'Historic Sheraton Palace Hotel, è stato interrotto per tre volte dagli applausi della platea.*

I presenti non credevano ai propri occhi: programmi per Windows anche di sole quattro (ripeto: quattro*) righe; compilazione in pochi istanti di programmi di centinaia di righe, risorse comprese; possibilità di portare sotto Windows un programma scritto con il Turbo Pascal per DOS solo cambiando «uses Crt» in «uses WinCrt»; debugging simbolico anche su un solo monitor; una gerarchia di classi* ObjectWindows*, analoga al Turbo Vision, costruita intorno alla API di Windows in modo tale da semplificarne enormemente l'uso. In breve, finalmente la programmazione sotto Windows alla portata di tutti, anche ... dei pigri: chiunque sappia programmare in Turbo Pascal sa già programmare sotto Windows; chi abbia lavorato con il Turbo Vision non impiegherà molto a padroneggiare* ObjectWindows*, e quindi a scrivere con poco sforzo applicazioni anche sofisticate.*

Non ci sono dubbi: non si può guardare al nuovo Pascal della Borland senza restarne affascinati. Se a San Francisco ho assistito agli applausi, poco più di un mese prima, a Milano, avevo potuto registrare l'entusiasmo con cui perfino Tommaso Masi aveva accolto un'analoga presentazione, lui strenuo difensore dello Smalltalk anche per Windows (perdonami Tommaso: dovere di cronaca!). Ne segue che il «giudizio» sul prodotto è presto dato, e che quindi questa prova non potrà limitarsi ad un normale esame: cercheremo di vedere se la enorme semplificazione di una programmazione tradizionalmente ostica nasconda qualche limitazione; cercheremo di capire le implicazioni di un prodotto indubbiamente rivoluzionario, anche con riguardo ad un rapporto Borland-Microsoft che, come vedremo, non è più di pura e semplice concorrenza.

Turbo Pascal per Windows

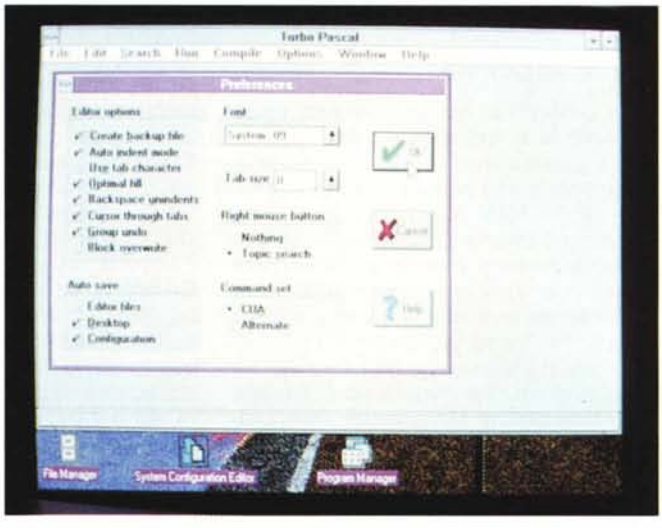
Produttore:
Borland International, Inc.
1800 Green Hills Road
P.O. Box 660001
Scotts Valley, CA 95067-0001
Distributore:
Borland Italia S.r.l.
Via Cavalcanti, 5 - 20127 Milano
Telefono: 02-2610102
Prezzi *(IVA esclusa):*
Turbo Pascal per Windows *L. 499.000*
Upgrade da qualsiasi linguaggio Borland *L.299.000*

Installazione

Quando qualcuno mi chiede che macchina comprare, rispondo sempre: almeno 80386SX, almeno due mega di RAM e una scheda madre su cui sia possibile installarne almeno quattro (meglio se otto), un monitor a colori almeno VGA, un disco rigido da almeno 40 mega con un tempo medio d'accesso non superiore a 25 millisecondi, un mouse. Non costa pochissimo, ma almeno non ti pentirai dell'acquisto dopo pochi mesi.

Il motivo per cui rispondo così è semplice: Windows. Alla Borland Languages Conference di San Francisco c'era pure Brad Silverberg, Vice Presidente della Microsoft responsabile della Systems Division. Ha parlato del futuro del DOS e di Windows, per dire che la Microsoft vede il DOS come sistema soprattutto per l'«utente casuale», quello che usa il PC sporadicamente e magari solo o soprattutto per qualche giochino. Per gli altri c'è e ci sarà sempre più Windows. È imminente una versione 3.1, con un nuovo *File Manager*, con i font scalabili e altre novità; sarà seguito a ruota da un Windows a 32 bit, in grado finalmente di sfruttare appieno le potenzialità dei processori 80386 e 80486 e con una API (*Application Programming Interface*) sostanzialmente identica a quella di Windows 3.0, al punto che sarà sufficiente apportare modifiche solo marginali ai programmi; il futuro sarà un sistema con nucleo NT (*New Technology*) su cui potranno girare applicazioni DOS, Windows, OS/2 e Posix (cioè Unix). Silverberg ne ha concluso (citando Philippe Kahn!), che chi voglia guardare lontano, chi voglia lunga vita per i propri programmi, già da oggi deve sviluppare sotto Windows.

Windows 3.0 può girare su qualsiasi macchina, ma è francamente ben poco attraente su un 8086, ed anche su un 80286 con meno di due mega di RAM non è esaltante. Va tuttavia rilevato che la RAM costa ora molto meno di un tempo, e magari anche che (come a San Francisco ha ricordato Andrew S. Grove, presidente della Intel) le vendite degli 80386 hanno superato quelle degli 80286 fin dall'aprile dello scorso anno, vale a dire un mese dopo l'annuncio di Windows 3.0. Il Turbo Pascal per Windows (TPW), per così dire, si adegua: può girare solo in modo protetto, richiedendo almeno un 80286 e almeno due mega di RAM. È la prima limitazione che dobbiamo registrare, ma, per quan-

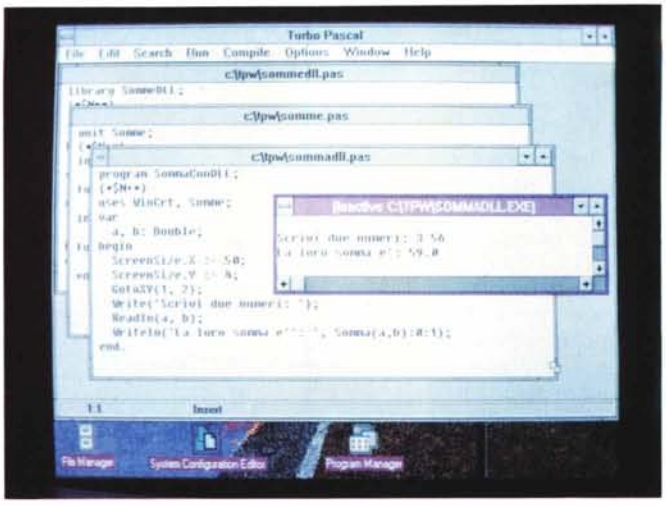


La dialog box della opzione Preferences *del menu* Options.

to appena detto, è certamente ragionevole.

Se si sceglie di installare anche tutti i programmi di esempio, ne risulta un'occupazione su disco di circa sei mega; si tratta di una scelta sicuramente consigliabile, dal momento che i vari «demo» aiutano molto a familiarizzare non solo con una programmazione sotto Windows di stile tradizionale (che rimane sempre possibile), ma anche e soprattutto con il ben più comodo approccio consentito dalla OOP e in particolare dalle gerarchie di classi *WObjects* e *ObjectWindows*.

Lo spazio su disco viene occupato non solo dal compilatore e dagli esempi, ma anche da un ampio help in linea (quasi due mega e mezzo) e da numerosi programmi accessori: dal Turbo Debugger per Windows ai *resource* e *Help Compiler* della Microsoft, dal *Whitewater Resource Toolkit* ai familiari MAKE, GREP e TOUCH, i quali tutti possono essere ovviamente cancellati se si dispone già di quelli forniti insieme al Borland C ++. È da notare che gli «accessori» sono tali che non c'è alcun bisogno di acquistare il *Software Development Kit* della Microsoft, e a ciò concorrono anche prodotti che la stessa Microsoft ha consentito venissero inclusi nella confezione. Per un semplice motivo: Windows, la cui diffusione non potrà che trarre beneficio dalla disponibilità di validi strumenti di sviluppo. E poco importa che questi siano targati Borland o altro. Tanta è la priorità che la Microsoft assegna alla diffusione di Windows, che sta anzi collaborando con la Borland per renderlo più object-oriented e per dotarlo di un migliore supporto per il debugging (così Gene Wang a Corrado Giustozzi e a me il 27 marzo a Milano). Collaborazione forse agevolata dal fatto che Silverberg, prima di passare alla Microsoft, ha lavorato alla Borland...



L'output di un breve programma scritto usando la unit WinCRT e che comprende tre file: SOMMEDLL.PAS (DLL), SOMME.PAS (unit di interfaccia alla DLL) e SOMMADLL.PAS (il programma).

L'output di un programma di 31 righe in cui si combinano la semplicità della unit WinCRT, la OOP (con la classe TStrCollection) e l'uso diretto di funzioni della API di Windows, per ottenere un elenco in ordine alfabetico delle intestazioni delle finestre delle applicazioni attive.

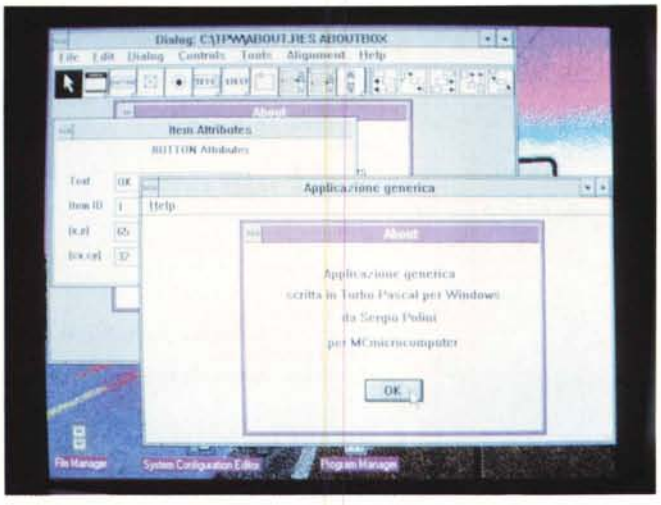
Documentazione

La confezione comprende ben sette manuali: la *User's Guide* e la *Programmer's Guide* sono molto simili agli analoghi testi forniti con il Pascal per DOS, mentre del tutto nuovi sono la *Windows Programming Guide* (dedicata alla programmazione mediante *ObjectWindows*) e la *Windows Reference Guide* (contenente un'analitica esposizione sia della API di Windows sia della gerarchia di *ObjectWindows*); gli altri tre manuali sono dedicati al nuovo *Turbo Debugger per Windows*, al *Whitewater Resource Toolkit* e all'*Help Compiler* della Microsoft.

A San Francisco ho chiesto a Richard Schell, Vice Presidente della Borland preposto allo sviluppo dei linguaggi, perché abbiamo due distinte versioni del Turbo Pascal, per DOS e per Windows, e se avremo in futuro un «Borland Pascal» unico per i due ambienti come già accade per C e C ++. Mi ha risposto che al momento non è prevista una versione bivalente del Pascal in quanto, mentre il Borland C ++ è concepito come prodotto completo per lo sviluppo professionale di applicazioni anche pesanti e complesse, il Turbo Pascal viene considerato un prodotto «d'entrata». Tale atteggiamento si comprende bene se si pensa che negli Stati Uniti il C/C ++ si giova della enorme diffusione che Unix vanta nelle università, ma in Europa la situazione è un po' diversa: in Germania, ad esempio, è proprio il Turbo Pascal il linguaggio più diffuso, seguito dal dBASE, dal COBOL e, solo quarto, dal C. In ogni caso, da quella impostazione segue che la Borland ha dedicato la massima attenzione a rendere agevole l'approccio al nuovo compilatore; al tempo stesso, si comprendono meglio alcune caratteristiche della documentazione. Intendo riferirmi in particolare ai due manuali dedicati alla programmazione sotto Windows.

La *Windows Programming Guide* segue la traccia di *ObjectWindows* per guidare passo passo il lettore nella realizzazione dei suoi primi programmi (gli esempi del manuale si ritrovano dopo l'installazione in una subdirectory DOC-DEMOS); la complessità della programmazione sotto Windows viene «nascosta» dietro una gerarchia di classi analoga — per quanto possibile — a quella del Turbo Vision, senza trascurare nulla: dalla interfaccia MDI alla gestione della memoria, dalle librerie DLL (*Dynamic-Link Libraries*) alla comunicazione tra applicazioni mediante DDE (*Dynamic Data Exchange*), dalla grafica con GDI (*Graphics Device Interface*) alle risorse (menu, acceleratori, dialog box, cursori, icone, stringhe, bitmap). Un «tutorial» completo e ben fatto, nel quale ho trovato pochissimi punti deboli.

Quando si tratta ad esempio delle funzioni *callback* (quelle chiamate da Windows invece che dall'applicazione), si dimentica di precisare che devono es-



L'output di una traduzione in ObjectWindows *del programma GENERIC, primo esempio di applicazione Windows proposto dai manuali dell'SDK.*

Il Whitewater Resource Toolkit *mentre viene usato per tradurre in italiano i menu del Turbo Pascal per Windows. Notate che è possibile verificare in ogni momento gli effetti delle modifiche apportate.*

sere «esportate» con la nuova keyword **export** e che, quando si passa il loro indirizzo ad un'altra funzione, questo deve essere ottenuto chiamando la funzione *MakeProcInstance* della API di Windows. Analogamente, la *Windows Reference Guide* propone una documentazione della API simile a quella — non esaltante — contenuta nel *Reference* dell'SDK della Microsoft, con qualcosa in meno: manca una esposizione sintetica delle oltre 500 funzioni raggruppate in categorie logiche (come invece si trova nelle prime 70 pagine del *Reference*), le descrizioni sono ancora più stringate.

In sintesi, la documentazione è di validissimo ausilio per moltissimi tipi di applicazioni, soprattutto se realizzate mediante *ObjectWindows*. Se tuttavia si vuole spremere Windows fino all'ultima funzione, è necessario munirsi di ulteriore supporto. Come d'altra parte capita anche (e anche più) a chi usi l'SDK: praticamente obbligatorio un testo come quello di Charles Petzold (*Programming Windows*, Microsoft Press, seconda edizione). È vero che si tratta di testi in cui si fa riferimento alla programmazione in C, ma questo potrebbe anche non essere un problema...

Implementazione del linguaggio

Nom mi soffermo sull'ambiente di sviluppo, in quanto si tratta di una normale applicazione Windows con interfaccia MDI, i cui menu sono molto simili a quelli del Pascal per DOS. Da segnalare, tuttavia, la possibilità di passare con un click del mouse dallo standard CUA di default ad una impostazione alternativa che ripropone l'interfaccia dei compilatori Borland per DOS (uscita con Alt-X invece che Alt-F4, Ctrl-QF per il *Find*, ecc.). Ne risulta un ambiente comodo e versatile, ma anche incredibilmente veloce, grazie al fatto che la compilazione avviene tutta in RAM sfruttando le capacità del microprocessore in modo protetto.

Dietro tale elegante veste, si svolge un compito arduo. Windows si presta particolarmente alla programmazione in C, ma preferisce che le chiamate di funzione avvengano seguendo le convenzioni del Pascal (passaggio dei parametri nell'ordine in cui sono scritti, ripristino dello stack a cura della funzione chiamata invece che di quella chiamante). Programmando in C si deve quindi rubare un po' dal Pascal, per farci programmare in Pascal la Borland ha rubato un po' dal C.

Con il Pascal 6.0 è stata introdotta una direttiva $X che attiva una «sintassi estesa», grazie alla quale è possibile chiamare una funzione senza badare al suo risultato, come se fosse una procedura (ne trovate esempi nella rubrica Turbo Pascal). Quella stessa direttiva viene ora a sua volta estesa per consentire di manipolare stringhe e puntatori a caratteri come si fa in C. Definendo un array di caratteri in modo che l'indice del primo sia zero, il tipo che ne risulta è compatibile con i puntatori a carattere di tipo *PChar*; proprio come in C posso inizializzare un puntatore a carattere con una stringa, col TPW posso assegnare una stringa (anche costante) ad un *PChar*; come avviene in C per ogni puntatore, si può accedere ai singoli caratteri della stringa puntata da un *PChar* con la stessa notazione usata per gli array (ad esempio: P[4] per il quinto carattere; il primo carattere, quello con indice zero, non è un byte di lunghezza: la fine della stringa è indicata dal carattere con codice zero); come in C, possono essere applicati al tipo PChar gli operatori «+», «−», «>», «<», «>=» e «<=». La unit STRINGS ci mette a disposizione un'intera serie di funzioni per la manipolazione delle nuove stringhe, tanto da non far rimpiangere le quasi omonime funzioni care al programmatore C (*StrLen, StrCat, StrIComp*, ecc.); le tradizionali *Read* e *Write*, *Val*, *Assign* e *Rename*, in compenso, possono operare sia sulle vecchie che sulle nuove stringhe.

Accanto a qualche ritocco al tipo *boolean*, la sintassi estesa consente di avere accesso a tutta la API di Windows, i cui tipi e le cui funzioni si ritrovano interamente e fedelmente «tradotti» nelle unit WINTYPES e WINPROCS.

Ma la API non basterebbe. Per sfruttare tutte le possibilità della programmazione sotto Windows si usano normalmente file DEF, con i quali si precisano, tra l'altro, le caratteristiche dei segmenti per codice e dati (FIXED o MOVEABLE, DISCARDABLE, PRELOAD o LOADONCALL, NONE o SINGLE o MULTIPLE), le dimensioni dello stack e del *local heap* (quello compreso nel segmento dati di ogni applicazione, mentre con *global heap* si indica la memoria a disposizione di tutte le applicazioni), le funzioni da «esportare» (vanno esportate le procedure e funzioni *callba-*

```
program Somma;
(*$N+*)
uses WinCRT;
var
  a, b: Double;
begin
  ScreenSize.X := 50;
  ScreenSize.Y := 4;
  GotoXY(1, 2);
  Write('Scrivi due numeri: ');
  Readln(a, b);
  Writeln('La loro somma e'': ', a+b:0:1);
end.
```

▲ *Listato 1 - Un breve esempio di programma che usa la unit WinCRT.*

```
library SommeDLL;
(*$N+*)
function Somma(a, b: Double): Double; export;
begin
  Somma := a + b;
end;
exports
  Somma index 1;
begin
end.

unit Somme;
(*$N+*)
interface
function Somma(a, b: Double): Double;
implementation
function Somma; external 'SOMMEDLL' index 1;
end.

program SommaConDLL;
(*$N+*)
uses WinCrt, Somme;
var
  a, b: Double;
begin
  ScreenSize.X := 50;
  ScreenSize.Y := 4;
  GotoXY(1, 2);
  Write('Scrivi due numeri: ');
  Readln(a, b);
  Writeln('La loro somma e'': ', Somma(a,b):0:1);
end.
```

Listato 2 - La somma di due numeri ottenuta con una DLL e una unit di interfaccia a questa. ▶

ck e quelle di una DLL). Il Turbo Pascal semplifica il tutto introducendo una direttiva $C per modificare gli attributi dei segmenti codice (per default: MOVEABLE, PRELOAD e PERMANENT, cioè non DISCARDABLE) e modificando la direttiva $M in modo che si riferisca solo alle dimensioni dello stack e del *local heap*. Quanto alle «esportazioni», basta aggiungere la keyword **export** all'intestazione della funzione o procedura interessata; nelle DLL, simili alle **unit** ma introdotte dalla keyword **library**, occorre poi riepilogare funzioni e procedure esportate in una apposita sezione denominata **exports**, dove è possibile indicare accanto al nome di ognuna un numero intero opzionale preceduto da **index**. Diciamo subito che anche le **library** possono ovviamente essere «usate»; può anzi essere «usata» qualsiasi DLL di Windows, anche se non realizzata in Turbo Pascal: per usare in un programma una funzione contenuta in una DLL, occorre approntare una **unit** nella cui sezione **implementation** si aggiunga al nome della funzione la familiare keyword **external** seguita dal nome della DLL tra apici e, opzionalmente, da **name** e dal nome con il quale la funzione è presente nella DLL (se diverso da quello che preferiamo per il nostro programma) o da **index** e dall'intero corrispondente. Sia il nome che l'indice delle funzioni di una DLL possono essere trovati con il programma TDUMP, fornito insieme al Turbo Debugger.

Quanto ai segmenti di dati, i manuali Microsoft ci dicono che le opzioni NONE e SINGLE possono essere usate solo per le librerie, MULTIPLE, FIXED e MOVEABLE solo per le applicazioni. I manuali del Turbo Pascal nulla dicono circa NONE, SINGLE e MULTIPLE, ma credo di poter ritenere che, essendoci poco da scegliere, vengano gestite automaticamente dal compilatore. È detto chiaro, invece, che il segmento dati di un'applicazione o di una libreria può essere solo FIXED. Per dirla in breve ciò comporta che, quando Windows è in *real mode*, può capitare che il *local heap* non possa essere espanso oltre il limite dettato dalla direttiva $M (8192 byte per default), in quanto il segmento dati non può essere mosso. Abbiamo cioè trovato un'altra limitazione. In effetti ci viene proposto di barattare una maggiore semplicità (ed efficienza) del compilatore con la rinuncia ad una elasticità che potrebbe tornare utile solo nelle situazioni in cui Windows stesso non gira al meglio. Nessun problema, infatti, nei modi *standard* e *386 enhanced* (un segmento mantiene lo stesso «selettore» anche se viene spostato in memoria).

Un analogo compromesso ci viene proposto per l'allocazione dinamica. Qui non vi sono limitazioni (rimane la possibilità di usare tutte le routine della API), ma di particolari scelte circa i meccanismi tradizionali del Pascal: non ci sono più *Mark* e *Release*, e *New*, *Dispose*, *GetMem* e *FreeMem* operano solo sul *global heap*. La gestione della memoria dinamica con tali procedure è ottimizzata per contrastare lo spreco che deriverebbe dal loro uso con blocchi «piccoli» di memoria (ogni blocco allocato nel *global heap* si porta dietro un overhead di almeno 20 byte): in caso di richiesta di allocazione di un blocco «piccolo» ne viene in realtà allocato uno «grande», che viene poi utilizzato per successive sub-allocazioni di altri blocchi «piccoli», senza overhead. Il meccanismo può essere perfino regolato agendo sulle variabili *HeapLimit* (dimensione massima di un blocco «piccolo») e *HeapBlock* (dimensione del blocco «grande»). Va rammentato peraltro che tutti i blocchi allocati con *New* o *GetMem* sono FIXED; ne segue che, in *real mode*, non possono essere mossi per constrastare la frammentazione del *global heap;* se quindi si dovessero proprio fare i conti con il *real mode*, potrebbe convenire usare le funzioni della API.

Librerie

Chi non abbia esperienza della programmazione sotto Windows se la immaginerà magari ora estremamente complicata. Non avrebbe tutti i torti. La unit WINCRT ci propone tuttavia un approccio estremamente semplice al nuovo ambiente.

Immaginate un qualsiasi programma scritto sotto DOS con la unit CRT: accanto a procedure standard come *Read* e *Write*, possiamo spostare il cursore con *GotoXY* e ottenerne la posizione con *WhereX* e *WhereY*, verificare se è stato premuto un tasto con *KeyPressed*, leggere i caratteri digitati con *ReadKey*, abilitare o disabilitare il Ctrl-C e il Ctrl-Break agendo sulla variabile *CheckBreak*, ecc. Bene. Tutto questo si può tranquillamente rifare sotto Windows usando la unit WINCRT invece che CRT. In tal modo, alla prima istruzione *Read* o *Write* si apre automaticamente sullo schermo una finestra che emula un normale terminale in modo testo; per default vi sono 25 righe e 80 colonne, ma le dimensioni possono essere scelte a piacere assegnando altri valori ai campi X e Y della variabile *ScreenSize* (naturalmente prima di qualsiasi operazione di lettura/scrittura); basta che il prodotto del numero di righe per il numero di colonne non superi 65520. Se le righe sono più di 25 o le colonne

```
program ListApps;
uses
  WObjects, WInTypes, WinProcs, WinCRT, Strings;
var
  fpProc: TFarProc;
  CaptionList: PCollection;
function WhoAmI(TheHandle: HWnd; TheValue: Longint): Integer; export;
var
  Str: array[0..79] of char;
begin
  if GetWindowText(TheHandle, Str, 79) <> 0 then
    CaptionList^.Insert(StrNew(Str));
 WhoAmI := 1;
end;
procedure Print(C: PCollection);
  procedure PrintCaption(P: PChar); far;
  begin
    Writeln(' ', P);
  end;
begin
  C^.ForEach(@PrintCaption);
end;
begin
  CaptionList := New(PStrCollection, Init(10, 5));
  fpProc := MakeProcInstance(@WhoAmI, hInstance);
  EnumWindows(fpProc, 0);
  ScreenSize.Y := CaptionList^.Count + 2;
  InactiveTitle := 'Intestazioni delle finestre delle applicazioni attive';
  Writeln;
    Print(CaptionList);
end.
```

◀ *Listato 3 - Un programma che legge le intestazioni delle finestre delle applicazioni attive e le mostra poi in una finestra creata dalla unit WinCRT. Da notare l'uso della classe* TStrCollection *per contare le intestazioni e produrle poi in ordine alfabetico.*

più di 80, è possibile far scorrere la finestra sullo schermo virtuale così definito sia mediante apposite istruzioni (*ScrollTo, TrackCursor*), sia agendo sulle scroll bar al termine dell'esecuzione del programma: la finestra rimane infatti aperta a meno che non si termini con *DoneWinCRT*. L'unico limite è rappresentato dal fatto che con *ReadKey* si possono leggere solo normali caratteri ASCII, non anche tasti funzione o simili, in quanto gestiti direttamente da Windows. C'è invece una curiosa inesattezza nell'help in linea: vi si legge che *DoneWinCrt* si limita a chiudere la finestra, lasciando la possibilità di riaprirne poi un'altra con *InitWinCrt*. In realtà, come ho detto, *DoneWinCrt* causa sempre la fine del programma. A San Francisco ho segnalato la cosa ad Anders, che mi ha detto che in effetti il comportamento descritto nell'help in linea è quello che avrebbe voluto implementare, ma che ha dovuto rinunciare perché... a Windows non piaceva. Mi ha anche assicurato che provvederà quanto prima a correggere testo ed esempi dell'help.

Anche il Turbo Pascal per Windows è comunque in primo luogo un linguaggio *object-oriented*, e la potenza che ne segue emerge tutta nella unit *ObjectWindows*.

L'incapsulamento (il codice insieme ai dati nella definizione di un tipo) semplifica l'accesso alle quasi 600 funzioni della API, diverse delle quali richiedono quattro o più parametri, mediante campi-dati corrispondenti a molti di tali parametri: l'assegnazione a loro di valori corretti avviene così in modo trasparente e con minori possibilità d'errore. Alcuni metodi, inoltre, svolgono compiti che nella API richiedono la chiamata di più funzioni, proponendo così un'interfaccia di più alto livello alla API stessa, senza limitare in alcun modo l'eventuale accesso diretto alle funzioni di questa. Ereditarietà e polimorfismo consentono di organizzare gli «oggetti» di Windows (finestre, menu, dialog box, scroll bar, ecc.) in una gerarchia molto ordinata, in cui vengono gestiti automaticamente, senza bisogno di lunghe esplicite e pesanti assegnazioni, i valori di default dei diversi attributi; per creare «oggetti» con caratteristiche diverse da quelle standard, è sufficiente inoltre derivare una nuova classe specificando solo quanto vi è di diverso da quella presa come base. Analogamente, la classe *TApplication* (un nome familiare per chi conosca il Turbo Vision) si prende cura automaticamente dietro le quinte di quelle operazioni per l'inizializzazione, il polling dei messaggi e l'uscita dall'applicazione che, praticamente uguali in quasi ogni programma Windows, si è invece costretti a ripetere esplicitamente ogni volta in altri ambienti (tipicamente il C con SDK).

Ciò che tuttavia davvero incanta è la gestione dei messaggi. Le classi *TWindowsObject e TWindow* (da quella derivata) definiscono metodi atti a rispondere a tutti i messaggi più frequenti che Windows può inviare ad un'applicazione (*WMCreate* per WM_CREATE, *WMPaint* per WM_PAINT, *WMLButtonDown* per *WM_LBUTTONDOWN*, e così via), i quali vengono chiamati *automaticamente* quando l'applicazione riceve il messaggio che ad ognuno compete. Lo stesso automatismo possiamo ottenere per qualsiasi altro messaggio, adottando una sintassi estesa per i metodi virtuali: basta aggiungere una costante dopo la keyword **virtual**. Si usano in pratica costanti simboliche predefinite per i messaggi di Windows (le stesse della API) e definite dal programmatore per gli altri, compresi quelli inviati a seguito della scelta di un'opzione di un menu o della interazione con una dialog box. Ne segue che spariscono del tutto le *window function* e le loro lunghissime istruzioni **switch** (l'equivalente in C del **case** del Pascal); ne segue che, se voglio che la mia applicazione risponda ad un nuovo messaggio, è sufficiente definire un metodo che si incarichi della cosa, con ovvi notevoli effetti sulla semplicità del programma e sulla facilità di manutenzione. Tanto per dare un'idea, il file GENERIC.C, che i manuali dell'SDK propongono come scarno e semplice esempio di una applicazione Windows minimale, conta 116 righe di codice (commenti esclusi!); il suo equivalente in *ObjectWindows* ne conta 34.

Quella variante sintattica comporta altri benefici. Le classi di *ObjectWindows* sono ricche di metodi e, con l'implementazione delle *Virtual Method Table* (VMT) che il Pascal per DOS ha mutuato dal C++ (e che rimane anche nel Pascal per Windows), la derivazione di classi comporta una certa occupazione di memoria, in quanto in ogni *table* vengono replicati gli indirizzi di tutti i metodi virtuali, anche di quelli che non vengono ridefiniti. Se invece si aggiunge una costante alla keyword **virtual**, il compilatore crea altre strutture, chiamate *Dynamic Method Table* (DMT), contenenti solo gli indirizzi dei metodi ridefiniti. Per quelli non ridefiniti si effettua una ricerca a ritroso nelle DMT delle classi da cui si è derivato. Un ovvio risparmio di spazio al prezzo di qualche ciclo del microprocessore.

Risorse

Spariscono dunque i file DEF, spariscono le *window function* e le loro lunghe istruzioni **switch**, ma spariscono anche i file RC. Insieme al Turbo Pascal per Windows viene «regalato» infatti il *Whitewater Resource Toolkit* (da solo costa 195 dollari), un eccellente strumento per la preparazione e la modifica

«dal vivo» di ogni tipo di risorse: acceleratori (combinazioni di tasti per attivare comandi normalmente accessibili scegliendo l'opzione di un menu), bitmap, cursori di varia foggia, icone, dialog box, menu, stringhe (i testi di una dialog box, i messaggi d'errore, ecc.).

Si tratta di uno strumento estremamente potente, tanto che può dare un'impressione di eccessiva complessità a chi vi si accosti per la prima volta. Ma si tratta solo di un'impressione: le diverse risorse possono essere composte sul video più o meno come si riempie una scheda Paradox (menu e stringhe) o come si disegna con PaintBrush, e possono poi essere salvate in molteplici formati; in particolare, possono essere creati direttamente i file RES, senza passare per i file RC e per il *Resource Compiler* della Microsoft. Dopo aver così creato le risorse di un'applicazione, basta indicare il nome del file RES come argomento di una direttiva $R perché il compilatore produca l'eseguibile per Windows, anche qui senza bisogno del *Resource Compiler*. Ne risulta una comodità tale che non si riesce a fare a meno del Toolkit e ci si familiarizza quindi ben presto con le sue numerose opzioni e con le diverse modalità d'uso.

Si può lamentare solo la concisione della documentazione cartacea, non riguardo al funzionamento del Toolkit, ma piuttosto circa le diverse possibilità che Windows ci mette a disposizione quando si tratta di definire le caratteristiche delle risorse: il manuale non riesce a dirci tutto quanto vorremmo sapere nelle sue 130 pagine, rimandando più volte al testo di Petzold che vi ho già citato. Potremmo considerare che si tratta di un testo da tutti riconosciuto come fondamentale, anche (e soprattutto) per chi usi l'SDK della Microsoft; il problema potrebbe essere che, per quanto mi risulta, non ne esiste ancora una traduzione italiana.

In compenso, col programma della Whitewater si può andare ben al di là della semplice produzione di risorse: non solo è possibile creare librerie DLL di sole risorse, o comporre in un file RES pronto per l'uso preesistenti file ICO, CUR e BMP (eventualmente, non necessariamente, creati con lo stesso Toolkit); è possibile anche estrarre risorse da un file EXE o DLL per poterne fare uso in altri programmi o librerie, o magari per modificarle e poi rimetterle nello stesso file da cui erano state estratte senza bisogno di ricompilarlo. Tanto per dare un'idea, ci si può fare in casa la versione italiana del Turbo Pascal per Windows traducendo i menu, i messaggi e i testi delle dialog box.

Segnalo infine che il Toolkit può essere usato anche in modo tradizionale, producendo o usando file H, DLG e RC da compilare con il *Resource Compiler* della Microsoft.

```
program About;
(*$R ABOUT.RES*)
uses WinTypes, WinProcs, WObjects;
const
  cm_About = 1;
type
  PMyWindow = ^TMyWindow;
  TMyWindow = object(TWindow)
    constructor Init(AParent: PWindowsObject; ATitle: PChar);
    procedure CMAbout(var Msg: TMessage); virtual cm_First+cm_About;
  end;
  TGenericApplication = object(TApplication)
    procedure InitMainWindow; virtual;
  end;
constructor TMyWindow.Init(AParent: PWindowsObject; ATitle: PChar);
begin
  TWindow.Init(AParent, ATitle);
  Attr.Menu := LoadMenu(Hinstance, 'ABOUTMENU');
end;
procedure TMyWindow.CMAbout(var Msg: TMessage);
begin
  Application^.ExecDialog(New(PDialog, Init(@Self, 'ABOUTBOX')));
end;
procedure TGenericApplication.InitMainWindow;
begin
  MainWindow := New(PMyWindow, Init(nil, 'Applicazione generica'));
end;
var
  MyApp: TGenericApplication;
begin
  MyApp.Init('Generic');
  MyApp.Run;
  MyApp.Done;
end.
```

Listato 4 - Traduzione in ObjectWindows *del programma GENERIC proposto nei manuali dell'SDK come primo esempio di applicazione Windows.*

Conclusioni

Manca lo spazio per una esposizione adeguata sia del nuovo Turbo Debugger che dell'*Help Compiler* della Microsoft, su cui torneremo magari in un'altra occasione. Mi limito a ricordare che il debugger può essere usato anche con un solo monitor, e che l'*Help Compiler* consente di dotare i nostri programmi di un help in linea identico a quello tipico di qualsiasi applicazione Windows.

Riassumendo quindi quanto fin qui detto, potremmo rilevare qua e là qualche carenza nella documentazione o una non completissima implementazione, nel senso che il Turbo Pascal per Windows non è forse lo strumento ideale per realizzare programmi ingombranti destinati a girare in *real mode* insieme ad altre ingombranti applicazioni Windows. Come dire che si tratta di un prodotto quasi perfetto. Chiunque programmi sotto Windows, infatti, se vuole effettivamente padroneggiarne le notevoli potenzialità, non può fermarsi né alla documentazione di un compilatore né a quella di un SDK; a voler esser pignoli, neanche il testo di Petzold è esaustivo (appena accennata la tecnica del «window subclassing», nessun cenno al «window superclassing» né agli hook). Quanto poi al *real mode*, lo definirei come un residuo di un mondo che si allontana sempre più, essendo suo compito solo quello di non punire chi non abbia ancora potuto aggiornare il suo hardware o i suoi pacchetti concepiti per precedenti versioni di Windows.

Con Windows 3.0 siamo finalmente entrati in una nuova fase, quella dell'ampia diffusione di ambienti operativi conformi ad uno standard intelligente e rigoroso (SAA-CUA) e capaci di sfruttare le potenzialità dei microprocessori a 32 bit. Con il Turbo Pascal per Windows abbiamo finalmente a disposizione uno strumento che, usando al meglio le possibilità della OOP e accompagnandosi ad ottimi accessori, riesce a rendere semplice veloce e divertente una programmazione fin qui accessibile solo a pazienti certosini.

La qualità è tale che avrebbe poco senso giudicare in rapporto ad essa il prezzo: il prodotto «vale» ampiamente quelle 499.000 lire. Sottolineo tuttavia con favore il perdurare di quella politica di «upgrade ad ampio spettro» che avevo registrato a proposito del Turbo Pascal 6.0: chiunque abbia già un qualsiasi linguaggio Borland può godere di uno sconto sostanzioso. Come resistere alla tentazione?

MC

Per conoscere meglio il tuo **ATARI®**

calamus

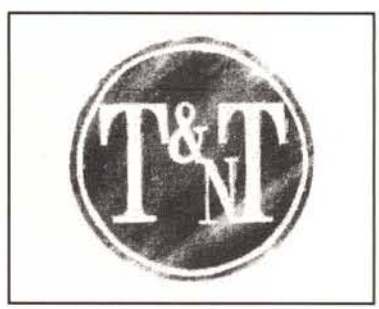
OUTLINE ART

Ditek® Dyna CADD
International

Scanners
Digitalizzatori
Stampanti
Plotters
Memory upgrade
Emulatori

Desk Top Publishing Center
Grafica Professionale
MIDI

STEFANO



HARDWARE
SOFTWARE

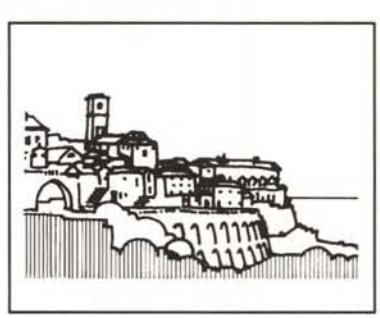
CORSI AVANZATI
STAGES
TOOLS PER:

DISEGNO TECNICO
GRAFICA CREATIVA
GRAFICA VETTORIALE
CAD
COLORS & B/W PROCESSING

via Santa Caterina da Siena, 38 - 66054 Vasto (CH)

Atari Club Vasto

Roland
DIGITAL GROUP



prefisso	0873
voce	365180
fax	362466
BBS	362472

ATARI®

Punto Consulenza Atari
Desk Top Publishing center