Turbo Pascal 5.5 e Quick Pascal 1.0

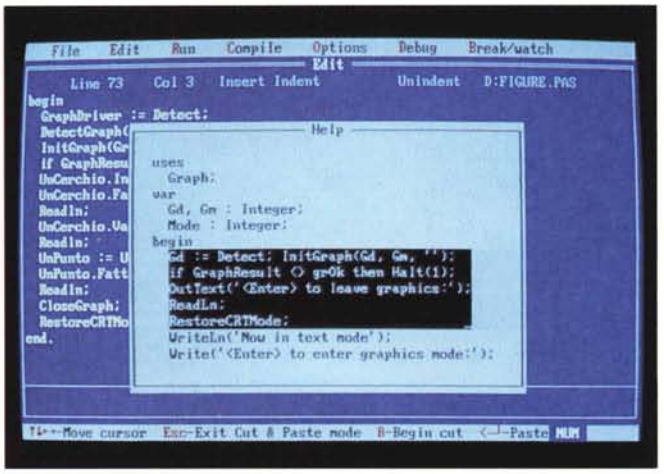
di Sergio Polini

Assistiamo a due eventi egualmente importanti: la Microsoft, abbandonando un pluriennale isolamento, ha finalmente riconosciuto che la Borland ha imposto un vero e proprio standard nel Pascal per PC; ha quindi offerto un proprio compilatore della serie Quick ampiamente compatibile con il Turbo Pascal 5.0. Sia la Borland che la Microsoft hanno poi dotato il Pascal di strumenti per la programmazione Object Oriented, ridando così nuova attualità ad un linguaggio che, dominatore dell'era della programmazione strutturata, rischiava di divenire in pochi anni troppo simile ad una mera testimonianza storica sotto i colpi di Ada, C, C++, LISP e PROLOG.

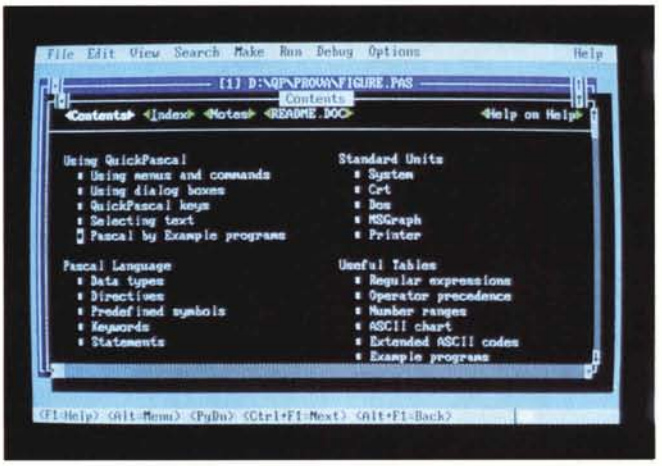
Non conosciamo tutti i dettagli di questo ennesimo confronto tra i due giganti del software. È certo che la Microsoft ha dovuto prendere atto dell'assoluto dominio della Borland nei compilatori Pascal, ha dovuto rassegnarsi a combatterla sul suo stesso terreno: quello di una implementazione tanto efficace quanto lontana dalla definizione ufficiale del linguaggio. Un commentatore americano, alle prese con il Pascal 4.0 della Microsoft, notava con una certa ironia che nel manuale, là dove si proponeva un confronto con altre implementazioni, il Turbo Pascal veniva semplicemente ignorato. Quasi fosse niente altro che stizza. In realtà si trattava di un atteggiamento almeno in parte giustificato. Fin dalla prova del Turbo Pascal 3.0 abbiamo sottolineato che il Pascal di Wirth, pur se eccellente per la didattica, era ben lontano dal possedere quel minimo di flessibilità che si richiede in applicazioni reali; abbiamo anche visto come l'implementazione della Borland fosse invece eccezionalmente fruibile: non solo un como-

dissimo ambiente di sviluppo interattivo, ma anche un Pascal flessibile e potente, comodo quasi quanto un BASIC interpretato, sbarazzino quasi quanto un C. Ne è risultato tuttavia anche un linguaggio ben lontano dallo standard, al punto che risulta proibitivo portare un programma Turbo Pascal in altri ambieti. Almeno in teoria, la Microsoft poteva dunque anche avere ragione; in pratica, però, centinaia di migliaia di utenti hanno dimostrato di preferire l'efficienza alla portabilità, e magari chi voleva la portabilità sceglieva semplicemente qualcos'altro, il C in primo luogo. La Microsoft, visto incrinato dal solito Turbo il suo dominio nei compilatori C, ha quindi finalmente accettato la sfida: non solo ha condiviso la nuova definizione del linguaggio imposta dalla Borland e dai suoi utenti, ma si è spinta oltre proponendo altre estensioni, ispirandosi all'Object Pascal sviluppato dalla Apple in collaborazione con papà Wirth.

È più difficile cercare di capire cosa è successo a Scotts Valley: le estensioni Object Oriented del Turbo Pascal derivano da un disegno da tempo tratteggiato, o sono solo la pronta risposta della Borland a quanto si sapeva stava armeggiando il concorrente? Il continuo «botta e risposta» tra i due farebbe propendere per la seconda ipotesi, ma va anche riconosciuto che le estensioni OOP della versione 5.5 non danno certo l'impressione della fretta: si presentano anzi più articolate e flessibili di quelle del Quick Pascal, meglio documentate, accompagnate da un migliore supporto. Soprattutto rispettano il tradizionale stile Borland: potenza e facilità d'uso preferite alla passiva adesione ad uno standard. Si trae infatti ispirazione dall'Object Pascal ma anche dal C++; si sostiene anzi che, così come il C si è evoluto nell'ANSI C e nel C++ facendo



L'help del Turbo Pascal 5.5. Le caratteristiche del linguaggio sono illustrate da brevi esempi. Premendo C si attiva il «cut & paste»: si posiziona il cursore e, dopo aver premuto B, si può marcare una zona rettangolare della finestra di Help. Premendo poi Enter quella zona viene copiata nella finestra di editing.



Premendo Alt-H si accede all'help del Quick Pascal. L'opzione Contents propone l'indice di quello che è un vero e proprio manuale in linea. Per accedere ai diversi «capitoli» si posiziona il cursore sul loro titolo e si preme Enter. In «Pascal by Example» si ritrovano gli esempi del manuale cartaceo, mentre «Example programs» ne propone altri. Con le opzioni «Cut» e «Paste» del menu di Edit si può trasferire in una finestra di editing ogni cosa dell'Help, esempi compresi.

L'opzione Index dell'help del Quick Pascal ne propone un indice analitico. Andando con il cursore sulla lettera H e premendo Enter compare l'indice per la lettera H; analogamente si procede poi alle schermate dedicate alle singole voci.

Turbo Pascal 5.5

Produttore:
Borland International, Inc.
1800 Green Hills Road
P.O. Box 660001
Scotts Valley, CA 95066-0001

Distributore:
EDIA Borland S.r.l.
Via Cavalcanti, 5 - 20127 Milano
Telefono: 02-2610102

Prezzi *(IVA esclusa):*

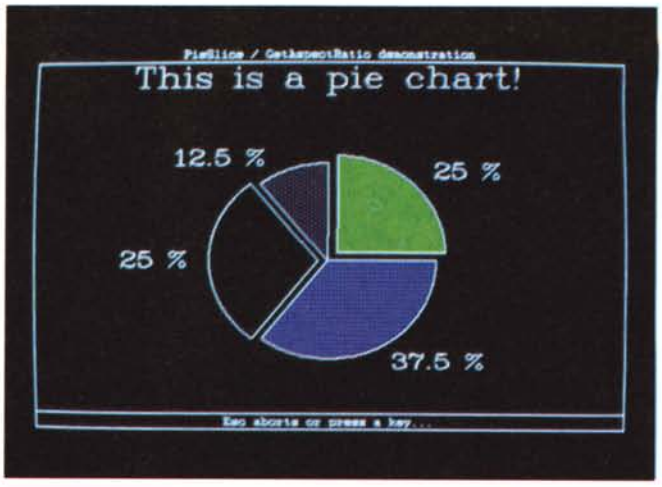
Turbo Pascal 5.5	*L. 299.000*
Turbo Assembler/Debugger 1.5	*L. 299.000*
Turbo Pascal 5.5 Professional	*L. 498.000*
Upgrade al 5.5	
— dalle versioni 3, 4 e 5.0	*L. 79.000*
Upgrade al 5.5 Professional	
— dalle versioni 3, 4 e 5.0	*L. 299.000*
— dal 5.0 Professional	*L. 199.000*

Turbo Pascal 1.0

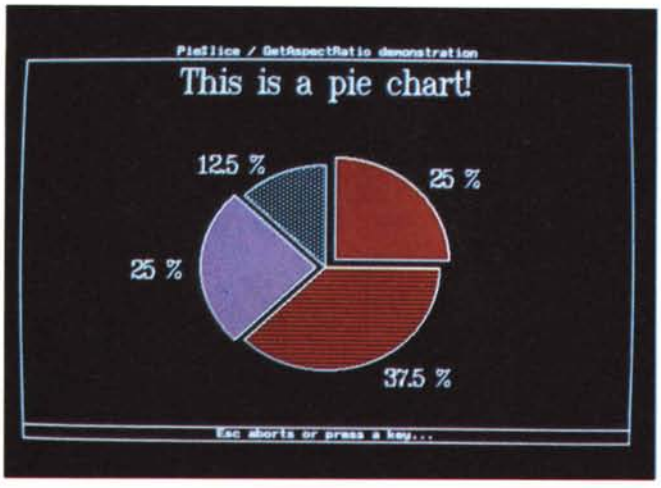
Produttore:
Microsoft Corporation
16011 NE 36th Way
Box 97017
Redmond, WA 98073-9717

Distributore:
Microsoft S.p.a.
Milano Oltre-palazzo Tiepolo
Via Cassanese 224, 20090 Segrate
Telefono: 02-2107201

Prezzo *(IVA esclusa):* *L. 195.000*



Il Quick Pascal è in grado di compilare il BGIDEMO del Turbo Pascal, ma con una resa dei colori a volte diversa.



Una schermata di BGIDEMO, il demo della Borland Graphics Interface

proprie anche molte caratteristiche tipiche del Pascal, questo deve evolversi traendo ispirazione dal C: le prime versioni Turbo ne avevano guadagnato i parametri variabile senza tipo, gli operatori di shift, le costanti tipizzate; la nuova versione offre metodi statici e virtuali, costruttori e distruttori mutuati dal C++. Viene quindi da pensare che anche sul terreno delle estensioni OOP sia stata la Microsoft a dover cercare di adattarsi alle scelte della Borland, e non viceversa.

Rimane certo, peraltro, che ambedue i prodotti propongono qualcosa di interessante, al punto che, quali che siano state la effettiva sequenza degli eventi e le loro motivazioni, noi utenti non potremo che trarre vantaggio dalla contesa.

Il Turbo Pascal 5.5, mantenendo totale compatibilità con la versione precedente (provata a gennaio), ne potenzia alcune caratteristiche e ne offre di radicalmente nuove: viene estesa la sintassi della istruzione **case**, *diventano più efficienti i meccanismi di overlay, migliora l'help in linea; vengono soprattutto aggiunte quattro keyword, con le quali si ha accesso a quello che la Borland giustamente definisce lo stile di programmazione degli anni '90, cioè a quella programmazione* object oriented *di cui abbiamo cercato di tratteggiare aspetti fondamentali e vantaggi nel numero di luglio/agosto.*

Il Quick Pascal 1.0 si propone come un'alternativa al TP 5.0: dichiara ampia compatibilità con questo e con la versione 4.0, offrendo anche una unit GRAPH che consente di compilare programmi scritti per la Borland Graphics Interface (BGI). Le unit MSGRAPH e MSGRUTIL propongono invece capacità grafiche analoghe a quelle del C 5.1 (quelle del QuickC 2.0 sono più potenti), e tre nuove keyword donano all'utente la possibilità di sviluppare programmi con una sintassi analoga a quella dell'Object Pascal del Macintosh. Il tutto in un ambiente di sviluppo interattivo stile Windows, mouse compreso.

Ambedue i compilatori sono dotati di un tutorial interattivo per aiutare l'utente a muovere i primi passi e di un compilatore separato per chi non gradisca l'ambiente integrato.

Installazione

Il TP 5.5 richiede almeno 384K di RAM, il QP 1.0 ne vuole 448; in entrambi i casi si tratta di minimi che conviene superare se si vogliono compilare programmi non brevi. Il Turbo Pascal è in grado di utilizzare la memoria EMS eventualmente presente per il buffer dell'editor (64K), il Quick Pascal può utilizzare un file di «swap» sia su EMS che su disco (si richiede tuttavia ben un Mega di spazio disponibile).

Ambedue i prodotti sono dotati di una elegante e facile procedura di installazione automatica, che guida l'utente sia nel caso disponga di un sistema a due floppy che di un hard disk (da considerare comunque raccomandabile). Il TP è dotato di un programma TINST che consente di variare sia la configurazione iniziale che altre caratteristiche, quali i comandi dell'editor.

Il QP provvede automaticamente a salvare in un file QP.INI alcune opzioni di configurazione, mentre un separato programma QPMKKEY permette di configurare l'editor integrato sul modello di Brief, Epsilon, EMACS, del Microsoft Editor oppure, infine, secondo il proprio gusto. La scelta della configurazione va data una prima volta sulla riga comando e viene poi registrata in QP.INI.

Dopo una installazione su disco rigido, ci si ritrova con circa 120 file e 1,7 Mega di disco occupati dal Turbo Pascal, circa 50 file e 1,6 Mega per il Quick Pascal (il quale potrebbe inoltre richiedere un ulteriore Mega per il file di «swap»).

L'occupazione su disco scende rapidamente con il Turbo Pascal se si eliminano o si «arcano» gli oltre 70 file di esempi, operazione meno utile con il Quick Pascal, in quanto la maggior parte degli esempi è incorporata nell'help del compilatore interattivo.

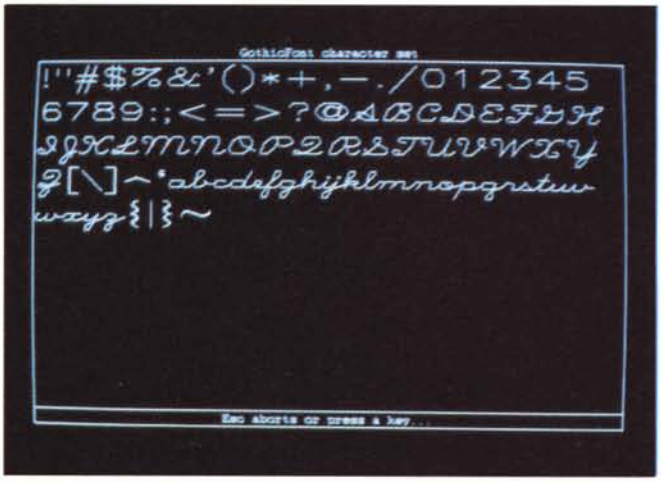
Documentazione

Abbiamo già descritto ampiamente, nei numeri di gennaio e febbraio, l'ottima documentazione che accompagnava il TP 5.0 Professional: 822 pagine per il compilatore e 1218 pagine per l'Assembler/Debugger (nella edizione originale), alle quali si aggiunge ora una guida alla programmazione object oriented di altre 118 pagine. Nei manuali si trovano efficaci tutorial, molti esempi, una completa guida di riferimento per tutte le caratteristiche del prodotto, dalla procedura d'installazione alle direttive di compilazione, dalla sintassi del linguaggio ai programmi accessori (MAKE, TOUCH, GREP, TPUMOVER, UPGRADE, ecc.), dalle funzioni e procedure delle varie unit a quei «segreti» della implementazione che possono mettere l'utente esperto in grado di portare a termine i progetti più «acrobatici». Il tutto disponibile anche in italiano.

Rapido ed efficace anche l'help in linea: non pretende di sostituire la documentazione cartacea, ma risulta utile quando non si ricordano il nome o la

Un'altra schermata del BGIDEMO compilato con il Turbo Pascal. Qui si vede cosa intende la Borland per «gotico».

e proprio «kolossal», molto simile a quello del QuickC 2.0. Vi sono tre tipi di aiuto: menu, linguaggio e generale (le definizioni sono nostre). Nel primo caso, premendo F1 si apre una finestra con la spiegazione delle diverse opzioni; nel



Ancora il BGIDEMO compilato con il Quick Pascal. Il «gotico» è in realtà uno script (i font del QP sono gli stessi di Windows).

sintassi di una qualche procedura, i dettagli delle direttive di compilazione, il contenuto delle varie unit, il significato delle opzioni dei menu, o magari i comandi dell'editor. Le novità della versione 5.5 sono due: in molti quadri è stata aggiunta una opzione *Examples* che conduce ad una esemplificazione dell'uso di ciò su cui si sta chiedendo aiuto, ed è possibile operare un *cut and paste* per trasferire il contenuto dell'esempio (come ogni altra cosa delle schermate di help) nel proprio sorgente. Come nella versione precedente, inoltre, è presente un programma residente THELP: una volta partito, basta poi premere il tasto che si è scelto (anche quei due strani oggetti targati F11 e F12) per attivare «da dovunque» l'help del compilatore. In altri termini, si dispone dell'help dell'ambiente integrato anche se si sceglie di usare un proprio editor e il compilatore separato (TPC.EXE). La nuova versione di THELP consente perfino un *cut and paste* analogo a quello on-line.

Se gli esempi dell'help sono piuttosto scarni, quelli nei file su disco sono quanto di meglio si possa desiderare. Nella versione 5.5 fanno spicco WINDEMO, un efficace esempio di manipolazione di uno stack di finestre sul video, la unit OBJECTS, su cui ritorneremo, e TCALC, una versione notevolmente potenziata dello spreadsheet che tradizionalmente accompagna i compilatori Borland: questa volta però si tratta di un programma object oriented in grado di gestire fino a cinque «quadranti», ognuno di 65535 righe per 65535 colonne. Un sostanzioso, concreto, utile esempio di OOP, con più di 8000 linee di sorgente (unit OBJECTS esclusa). Elementare e incompleto invece il tutorial interattivo: offre i primi rudimenti a chi muova i primi passi nell'ambiente integrato, ma nulla dice circa il debugging.

Completamente diverso l'approccio Microsoft. Si sa che i Quick sono la versione per prototyping, o per l'utente casuale o principiante, dei compilatori «seri», e il Quick Pascal sembra seguire una identica filosofia; manca però il Pascal «serio». Il manualetto *Up and Running* (62 pagine) guida nella fase d'installazione e illustra sufficientemente l'uso dell'ambiente integrato, editor e help compresi (a proposito: ma perché per uscire dall'help si deve premere un innaturale Ctrl-F4?). Decisamente carenti invece l'illustrazione del compilatore separato (meno di due pagine) e del debugger (meno di una pagina). Quanto al primo soccorre un «file» README.DOC, accessibile dall'help in linea ma diverso dal file omonimo su disco, quanto al secondo ci sono solo il tutorial interattivo (più completo quindi di quello della Borland, ma ugualmente elementare) e l'help in linea. Il secondo manuale, *Pascal by Example* (289 pagine), è quello che potete indovinare dal titolo: un corso sulla programmazione in Pascal. Da tale punto di vista merita un bel 10, in quanto sicuramente utilissimo per i principianti grazie alla gradualità e alla chiarezza della esposizione. Non è però un *reference*, e si vede: è tutt'altro che completo, le diverse unit e le direttive di compilazione sono descritte solo in veloci appendici (da integrare con l'help in linea); solo la grafica stile-Microsoft è sufficientemente illustrata, mentre sono decisamente troppo poche le 11 pagine dedicate alle estensioni object oriented.

Questo non vuol dire che l'utente sia lasciato a se stesso. Pur con alcune eccezioni, l'help in linea colma parecchie lacune dei manuali; è anzi un vero secondo, mentre si sta scrivendo o correggendo un sorgente, F1 offre spiegazioni sulla keyword o parola predefinita su cui è posizionato il cursore; nel terzo, che si attiva con Alt-H, si può accedere alle opzioni *Contents* e *Index*, che propongono diverse vie d'accesso alle numerose videate di quello che è una sorta di manuale interattivo, completo di 84 programmi esemplificativi.

Con il *cut and paste* si possono trasferire nella finestra di editing sia interi programmi che qualsiasi altra cosa contenuta nell'help. La navigazione è facile e comoda con il mouse, meno con la tastiera; in ogni caso si arriva quasi sempre in breve ad ottenere la risposta desiderata. «Quasi sempre» perché alcune cose (ad esempio la direttiva G) sono documentate solo nel «file» README.DOC interno all'help, che bisogna quindi ricordarsi di consultare.

Questo è un piccolo peccatuccio della Microsoft: se siamo ormai abituati (o meglio rassegnati) a file README che integrano e correggono i manuali, la documentazione on-line dovrebbe essere tutta aggiornata. È poi un po' incoerente costringere l'utente a cercare nell'help dell'ambiente integrato la documentazione del compilatore separato. In definitiva, tuttavia, si tratta soprattutto di una diversa filosofia: più carta (anche in italiano) e meno on-line la Borland, meno carta (in inglese) e più on-line la Microsoft. Sarà il mercato ad esprimere la sua preferenza, e probabilmente chi gradirà la filosofia Microsoft perdonerà

anche le attese che risultano talvolta necessaria perché il Quick Pascal trovi quello che gli chiediamo nei circa 450K dei cinque file di help su disco.

Ciò salomonicamente detto, dobbiamo tuttavia rilevare alcune singolari carenze. Manca in particolare qualsiasi cenno alla Borland Graphics Interface: si dichiara una compatibilità che l'utente deve verificare per tentativi, in quanto nulla si dice, né su carta né on-line, delle oltre 70 procedure e funzioni della BGI. Solo provando si può scoprire che font e colori hanno una resa diversa, e che non è possibile inserire i file con i font nel programma, come con il Turbo Pascal. Ed è un peccato, anche perché la grafica del Quick Pascal (nelle unit MSGRAPH e MSGRUTIL) è meno potente sia di quella del QuickC 2.0 che di quella Borland per applicazioni di *business graphics*. Vi sono poi caratteristiche del linguaggio menzionate di sfuggi-

```
Program Figure;
uses Graph, Crt;
type
  Punto = object
    X, Y: integer;
    procedure Init;
    procedure FattiVedere;
    procedure Sparisci;
    procedure VaiA(NuovoX, NuovoY: integer);
  end;
  Cerchio = object(Punto)        (* sottoclasse di Punto *)
    Raggio: integer;
    procedure Init; override;
    procedure FattiVedere; override;
    procedure Sparisci; override;
  end;          (* X, Y e VaiA sono "ereditati" da Punto *)

procedure Punto.Init;
begin
  Self.X := 150; Self.Y := 100;
end;
procedure Punto.FattiVedere;
begin
  PutPixel(Self.X, Self.Y, GetColor);
end;
procedure Punto.Sparisci;
begin
  PutPixel(Self.X, Self.Y, GetBkColor);
end;
procedure Punto.VaiA(NuovoX, NuovoY: integer);
begin
  Self.Sparisci;
  Self.X := NuovoX; Self.Y := NuovoY;
  Self.FattiVedere;
end;

procedure Cerchio.Init;
begin
  inherited Self.Init;
  Self.Raggio := 50;
end;
procedure Cerchio.FattiVedere;
begin
  Circle(Self.X, Self.Y, Self.Raggio);
end;
procedure Cerchio.Sparisci;
var
  Temp: word;
begin
  Temp := GetColor;
  SetColor(GetBkColor);
  Circle(Self.X, Self.Y, Self.Raggio);
  SetColor(Temp);
end;
var
  GraphDriver, GraphMode: integer;
  UnPunto: Punto;
  UnCerchio: Cerchio;
begin
  GraphDriver := Detect;
  DetectGraph(GraphDriver, GraphMode);
  InitGraph(GraphDriver, GraphMode, '..');
  if GraphResult <> GrOk then Halt(1);
  New(UnCerchio);
  UnCerchio.Init;          (* X: 150, Y: 100, Raggio: 50 *)
  UnCerchio.FattiVedere;
  Readln;
  UnCerchio.VaiA(300, 100);
  Readln;
  New(UnPunto);
  UnPunto.Init;                          (* X: 150, Y: 100 *)
  UnPunto := UnCerchio;
  UnPunto.FattiVedere;     (* chiama Cerchio.FattiVedere *)
  Readln;
  CloseGraph;
  RestoreCRTMode;
end.
```

```
Program Figure;
uses Graph, Crt;
type
  Punto = object
    X, Y: integer;
    constructor Init(InitX, InitY: integer);
    procedure FattiVedere; virtual;
    procedure Sparisci; virtual;
    procedure VaiA(NuovoX, NuovoY: integer);
  end;
  Cerchio = object(Punto)        (* sottoclasse di Punto *)
    Raggio: integer;
    constructor Init(InitX, InitY, InitR: integer);
    procedure FattiVedere; virtual;
    procedure Sparisci; virtual;
  end;          (* X, Y e VaiA sono "ereditati" da Punto *)

constructor Punto.Init(InitX, InitY: integer);
begin
  X := InitX; Y := InitY;
end;
procedure Punto.FattiVedere;
begin
  PutPixel(X, Y, GetColor);
end;
procedure Punto.Sparisci;
begin
  PutPixel(X, Y, GetBkColor);
end;
procedure Punto.VaiA(NuovoX, NuovoY: integer);
begin
  Sparisci; X := NuovoX; Y := NuovoY; FattiVedere;
end;

constructor Cerchio.Init(InitX, InitY, InitR: integer);
begin
  Punto.Init(InitX, InitY);
  Raggio := InitR;
end;
procedure Cerchio.FattiVedere;
begin
  Circle(X, Y, Raggio);
end;
procedure Cerchio.Sparisci;
var
  Temp: word;
begin
  Temp := GetColor;
  SetColor(GetBkColor);
  Circle(X, Y, Raggio);
  SetColor(Temp);
end;
const                        (* costante-oggetto tipizzata *)
  UnPunto: Punto = (X: 150; Y: 100);
var
  GraphDriver, GraphMode: integer;
  UnCerchio: Cerchio;
begin
  GraphDriver := Detect;
  DetectGraph(GraphDriver, GraphMode);
  InitGraph(GraphDriver, GraphMode, '..');
  if GraphResult <> GrOk then Halt(1);
  UnCerchio.Init(150, 100, 50);
  UnCerchio.FattiVedere;
  Readln;
  UnCerchio.VaiA(300, 100);
  Readln;
  UnPunto := UnCerchio;  (* effetto: X := 300; Y := 100 *)
  UnPunto.FattiVedere;       (* chiama Punto.FattiVedere *)
  Readln;
  CloseGraph;
  RestoreCRTMode;
end.
```

A sinistra un esempio di OOP in Quick Pascal. La classe Cerchio eredita così come sono i dati e il metodo VaiA della classe Punto da cui discende; ridefinisce invece gli altri metodi. Quando si chiama UnCerchio.VaiA, viene appunto chiamato il metodo Punto.VaiA, il quale chiama a sua volta Sparisci e FattiVedere; poiché questi sono «overridden», il codice che viene eseguito è quello di Cerchio.Sparisci e Cerchio.FattiVedere, nonostante che non venga esplicitamente definito alcun metodo Cerchio.VaiA. Gli «oggetti» UnPunto e UnCerchio rispondono ognuno «con i propri metodi» al «messaggio» VaiA(NuovoX, NuovoY).

A destra un esempio di OOP in Turbo Pascal. Vi sono sia metodi statici che metodi virtuali. L'inizializzazione degli oggetti avviene mediante metodi statici detti «costruttori» i quali, proprio perché statici, ammettono dichiarazioni diverse (quanto al numero e al tipo dei parametri) da oggetto a oggetto. È possibile sia dichiarare costanti-oggetto tipizzate che assegnare ad un oggetto di una classe un oggetto istanza di una classe discendente: l'effetto è quello di assegnare solo i campi che il secondo ha ereditato dal primo.

ta nel manuale ma assenti on-line (il tipo **pointer**), altre illustrate solo nell'help (le istruzioni e la direttiva **inline**, la keyword **interrupt**, le variabili **absolute**); manca del tutto la documentazione dei parametri variabile senza tipo e di parametri e variabili procedurali. Tutti aspetti cruciali per una piena compatibilità con il Turbo Pascal. In altri termini, tale compatibilità è dichiarata ma non documentata, ed è lasciato all'utente il compito di scoprire cosa il Quick Pascal accetta e non accetta (stando alle nostre prove, tutti gli aspetti ora menzionati sono «compatibili»). Manca infine l'accesso all'help «da fuori», manca cioè un omologo del THELP della Borland.

Ultima componente della documentazione sono i messaggi d'errore: non sempre il Quick Pascal è efficace come il Turbo Pascal. Nel corso della prova, ci è capitato di digitare un programma proposto dal manuale copiando distrattamente un banale errore: veniva chiamata una funzione senza nulla fare del suo risultato, come se fosse stata una procedura (l'errore, non corretto nei README, si trova in *Pascal by Example*, pagina 174, penultima riga del sorgente). Ne è seguito il messaggio *Invalid variable reference*. Mistero. Chiesto aiuto con F1, è apparso un messaggio secondo il quale veniva passato qualcosa di diverso da una variabile ad una procedura che aspettava un parametro variabile. Non c'entrava niente. Scoperta poi (per puro mestiere...) la vera natura dell'errore, abbiamo provato a proporne uno simile al Turbo Pascal: l'identico oscuro messaggio d'errore, ma un diverso help, secondo il quale «molto probabilmente» si stava cercando di usare una funzione senza «dereferenziarne» il risultato. Non proprio ideale quanto a chiarezza, ma più aderente alla effettiva natura del problema. Avremo modo di rilevare tra breve altri piccoli punti deboli nei messaggi d'errore del Quick Pascal.

Implementazione del linguaggio

Due compilatori non standard, ma conformi... allo standard Borland. Per quanto detto sopra non è agevole determinare quanto il Quick Pascal sia compatibile con il TP 5.0, ma qualcosa almeno è evidente: non è possibile compilare programmi che richiedano l'overlay o le unit TURBO3 e GRAPH3, colori e font della BGI hanno una resa diversa, non si possono incorporare i font nei file EXE, non si possono «linkare» file OBJ prodotti dal Turbo C (e nemmeno dal QuickC; solo Assembler). Un'altra curiosa differenza la troviamo nel trattamento dei numeri non interi. Nel Turbo Pascal si può agire su una direttiva N: se disattivata, è disponibile solo il tipo *real*, se attivata si dispone anche di *single*, *double*, *extended* e *comp*, gli ultimi due con 19-20 cifre significative; ciò richiede la presenza del coprocessore 80x87, ma attivando una direttiva E questo viene perfettamente emulato. Nel Quick Pascal la direttiva N agisce diversamente: se attivata richiede la presenza del coprocessore, altrimenti si dispone ugualmente di tutti i tipi, ma l'80x87 non viene emulato, con la conseguenza non solo che *extended* e *comp* non vanno oltre le 15-16 cifre significative, ma anche che il coprocessore non viene utilizzato pure se presente.

Quanto al resto, la compatibilità sembra effettivamente molto alta, con alcune piccole ma interessanti estensioni: l'istruzione case accetta selettori anche di tipo *longint*, le funzioni predefinite *First* e *Last* ritornano il primo e l'ultimo elemento di un tipo scalare, si può accedere ad una variabile dichiarata in un blocco esterno pure da un blocco in cui ne esista un'altra con lo stesso nome, premettendo il nome del blocco e un punto (se A1 è dichiarata in una procedura P1, e se P2, nidificata in P1, dichiara un'altra variabile A1, si può accedere alla prima con P1.A1); esiste infine un tipo CSTRING compatibile con il classico tipo STRING, ma senza un byte iniziale di lunghezza e con un byte zero alla fine: come dice il nome, le stringhe del C. Dal punto di vista delle novità «minori», la Borland ci concede invece solo una istruzione case con selettori di tipo *word*.

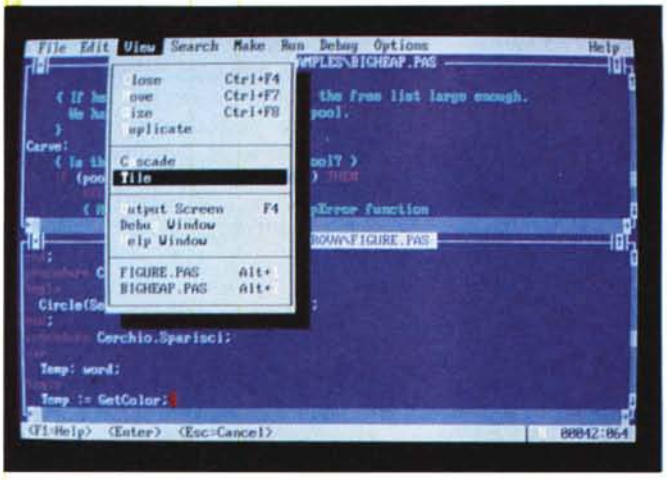
Le principali novità sono comunque rappresentate dalle estensioni object oriented: con pochissimi ritocchi al linguaggio, ambedue i compilatori consentono all'utente di affacciarsi in un mondo completamente nuovo.

Nel Quick Pascal abbiamo tre nuove keyword: **object**, **override** e **inherited**. Con la prima si definiscono classi di oggetti quasi come si definiscono i record: le due differenze sono che possono essere indicate funzioni e procedure (i «metodi») oltre ai campi di dati, e che può essere stabilita una discendenza da una classe precedente, nel senso che se ne ereditano dati e metodi senza bisogno di ripetere la dichiarazione. Se una sottoclasse vuole ridefinire un metodo della classe da cui discende, può ripeterne la dichiarazione aggiungendo la keyword **override**, che — parere del tutto personale — è in realtà superflua, in quanto non sono possibili alternative: o i nomi dei metodi sono diversi, o il metodo che si sovrappone a quello «ereditato» dalla superclasse deve essere dichiarato nello stesso identico modo (uguali il numero e il tipo dei parametri) e deve essere seguito da **override**. Se ci si dimentica della nuova keyword i messaggi d'errore e i relativi help non sono poi di grande aiuto: suggeriscono solo di cambiare nome al metodo.

Nella definizione di una classe va riportata solo l'intestazione dei metodi (quasi come si trattasse di dichiarazioni **forward**); quando poi si implementa un metodo, questo appare come una normale funzione o procedura; il nome va però preceduto dal nome della classe e da un punto, e l'accesso ai dati della classe è possibile solo mediante una pseudovariabile **Self**, che va trattata come se fosse una variabile di un tipo record eguale alla classe; in pratica, i metodi di una classe C possono accedere ad una variabile X di C con

```
type
  PuntoPtr = ^Punto;

var
  PPtr1, PPtr2, PPtr3, PPtr4: PuntoPtr;

begin
  New(PPtr1);                          (* sintassi tradizionale di New *)
  PPtr1^.Init(78, 187);                (* chiamata del constructor *)

  New(PPtr2, Init(137, 82));           (* il constructor come secondo argomento *

  PPtr3 := New(PuntoPtr);              (* New come funzione *)
  PPtr3^.Init(40, 200);

  PPtr4 := New(PuntoPtr, Init(10, 0));

  ...

  PPtr1^.Destruct;                     (* chiamata del destructor *)
  Dispose(PPtr1);                      (* sintassi tradizionale di Dispose *)

  Dispose(PPtr2, Destruct);            (* il destructor come secondo argomento *)
end.
```

Il Turbo Pascal estende la sintassi delle procedure New e Dispose per rendere più comodo l'uso di costruttori e distruttori. Una curiosità: le procedure GetMem e FreeMem del Turbo Pascal ricalcano le funzioni malloc() e free() del C; costruttori e distruttori derivano dal C++, il quale sostituisce a malloc() e free() due funzioni new() e delete() molto simili alle New e Dispose del Pascal.



Con il Quick Pascal si possono aprire più finestre di editing. Le opzioni Cascade e Tiled del menu View consentono di scegliere tra finestre l'una sull'altra (e di andare alla prima con Alt-1, alla seconda con Alt-2, ecc.) o accostate. Attivando una opzione Pascal Syntax dal menu Edit, vengono evidenziati in colori diversi le parole riservate, le stringhe e i commenti.

«Self.X». Analogamente per l'accesso agli altri metodi della stessa classe. Se la chiamata di un metodo dichiarato **override** è preceduta dalla keyword **inherited**, si chiama invece il metodo con lo stesso nome della classe immediatamente antecedente. È possibile definire una gerarchia di classi articolata su più livelli: da A si può derivare B, da B si può derivare C; gli oggetti «istanza» della classe C ereditano dati e metodi non «overridden» delle classi superiori, ma possono avere accesso ai metodi «overridden» solo della classe immediatamente superiore nella gerarchia: se si prova a chiamare un metodo «overridden» di A da un oggetto della classe C il compilatore si rifiuta, avvertendoci che ... abbiamo dimenticato una parentesi tonda!

Per «oggetti istanza di una classe» si intendono variabili il cui tipo sia una classe. Per creare tali variabili non basta però dichiararle, in quanto sono una sorta di puntatori. Se si dimentica di crearli con un *New* il crash è assicurato, e proprio per questo si dispone di una direttiva M che, se abilitata (come è per default), genera un errore di esecuzione se si accede ai metodi di un oggetto non allocato con *New*. La documentazione non dice se si può tranquillamente disattivare la direttiva dopo il debugging (la sua attivazione è invece sempre raccomandata), ma dalle nostre prime prove risulterebbe possibile. Tanto sembrano puntatori gli oggetti, che assegnando un oggetto della classe C ad uno della classe B questo quasi «diventa» un oggetto della classe C: se dichiaro oggetti appartenenti alle classi Punto e Cerchio (questa discendente da quella), se li alloco e li inizializzo, se poi assegno un Cerchio a un Punto e chiamo per il Punto il metodo FattiVedere comune alle due classi, vedo un cerchio (con il Turbo Pascal vedrei un punto nella posizione in cui sarebbe stato il centro del cerchio).

Per dirla col gergo del Turbo Pascal, i metodi del Quick sono tutti «virtuali»: si adattano cioè all'oggetto a cui vengono inviati (come «messaggi»), in quanto il codice che viene effettivamente eseguito con la chiamata del metodo viene deciso al momento della stessa esecuzione, non già durante la compilazione.

Nel Turbo abbiamo invece, oltre ad una simile keyword **object**, la possibilità di dichiarare **virtual** solo alcuni metodi, lasciando gli altri «statici» (il codice che verrà eseguito con la loro chiamata è determinato al momento della compilazione). Ciò consente un po' di efficienza in più, in quanto ci si può limitare ad usare i metodi virtuali solo dove effettivamente ce n'è bisogno. Non vi è necessità, inoltre, di allocare con *New* gli oggetti se non quando si vogliono adottare strutture di dati dinamiche; si comportano insomma come variabili normali (sono definibili anche costanti-oggetto tipizzate, impossibili in QP), con l'unica restrizione che bisogna chiamare un particolare tipo di metodo, detto **constructor**, per ogni variabile che sia istanza di una classe contenente metodi virtuali. Se non lo si fa il crash è assicurato, e proprio per questo è stata estesa la direttiva R che, se attivata, controlla anche che non si acceda a variabili del genere prima di chiamare un **constructor**. Dopo il debugging la direttiva può essere tranquillamente disattivata. Esiste anche nel Turbo Pascal un *Self*, il cui uso è però opzionale e destinato unicamente ad eliminare possibili ambiguità; si possono chiamare liberamente anche i metodi «overridden» delle classi superiori.

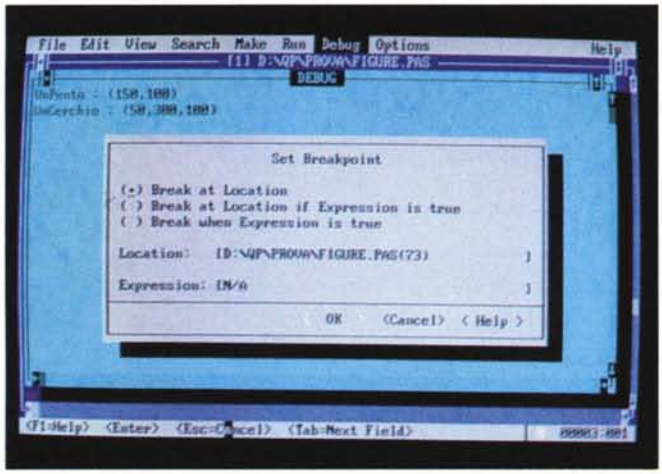
Se un metodo è statico (come devono essere i «costruttori»), una sottoclasse vi può sovrapporre propri metodi con lo stesso nome ma con dichiarazione anche diversa quanto al numero e al tipo dei parametri. Se è invece **virtual** non solo ciò non è possibile, ma la dichiarazione va ripetuta per tutte le classi discendenti, keyword **virtual** compresa. Anche questa — parere ancora del tutto personale — ci sembra una inutile ridondanza, tanto più che non è necessaria nel linguaggio da cui il Turbo Pascal ha tratto ispirazione: in C++ basta dichiarare come «virtuale» un metodo solo la prima volta, essendo poi scontato (appunto perché non vi sono alternative) che il metodo sarà virtuale in tutte le sottoclassi. Almeno se dimentichiamo la keyword il messaggio d'errore non lascia dubbi: «VIRTUAL expected».

La maggiore flessibilità di cui si gode nella dichiarazione dei metodi statici ha una utile conseguenza: è possibile definire per ogni classe un metodo di inizializzazione con un nome standard (la Borland usa *Init*), ma con numero e tipo di parametri adeguati ai vari oggetti: passerò le coordinate X e Y ad un punto, le coordinate e il raggio ad un cerchio, il numero degli elementi ad un array dinamico, l'elenco delle opzioni ad un menu, e così via, sempre con *Init*, senza bisogno di inventare per ogni classe un nome diverso per la procedura di inizializzazione. Non è proprio il *function overloading* del C++, ma almeno un po' gli assomiglia. Con il Quick Pascal ciò è impossibile, in quanto tutti i metodi sono virtuali; ne segue che o si inventano tanti nomi (ma non si saprebbe che fare delle procedure di inizializzazione delle classi superiori, che verrebbero comunque «ereditate»), o si fa come la Microsoft, che nei suoi esempi on- line usa tutti metodi *Init* senza parametri, limitandosi ad assegnare valori costanti alle variabili di classe (con ovvio pregiudizio della riusabilità del codice).

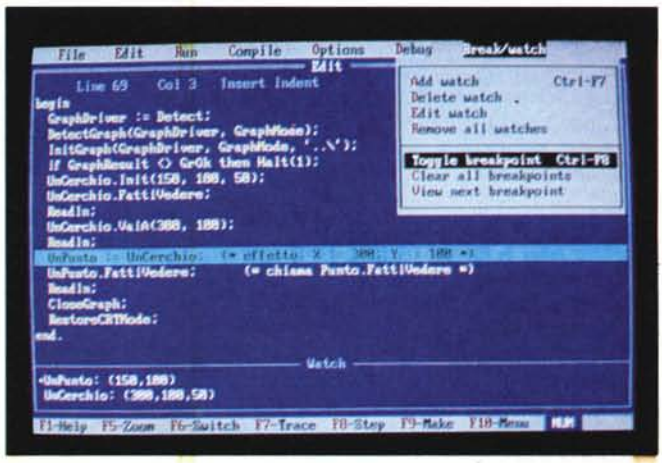
La quarta nuova keyword del Turbo Pascal è **destructor**. I «distruttori» sono necessari solo nel caso di oggetti allocati dinamicamente, in quanto guidano il processo di deallocazione della memoria passando alla procedura *Dispose* l'informazione relativa alla quantità di memoria occupata da un oggetto, non determinabile a priori per oggetti che usino metodi virtuali. La sintassi di *New* e *Dispose* è stata anzi estesa per consentire un più agile uso di puntatori, costruttori e distruttori con oggetti allocati dinamicamente.

La Guida OOP del Turbo Pascal spiega poi con dovizia di esempi la compatibilità tra oggetti appartenenti a classi diverse: si può assegnare ad un oggetto istanza di una classe anche un oggetto istanza di una classe da questa discendente (succede semplicemente che vengono assegnati solo i «campi» che il secondo ha ereditato dal primo), si può passare ad una procedura che vuole un parametro di una certa classe anche un oggetto istanza di una classe discendente. Il tutto senza che un oggetto «diventi» un altro.

Concludiamo segnalando che la Gui-



Il debugger integrato del Quick Pascal offre la possibilità di impostare dei breakpoint condizionati (scattano quando diventa vera una data espressione) e incondizionati (quando si raggiunge una data istruzione), nonché di tenere sotto costante osservazione il valore di alcune variabili.



Il debugger integrato del Turbo Pascal dispone di breakpoint incondizionati e di una finestra di «watch» per tenere sotto osservazione il valore di alcune variabili.

da illustra in dettaglio anche l'implementazione delle estensioni OOP, fino a proporre esempi di metodi scritti in Assembler.

Librerie

Già conosciamo le otto unit predefinite del Turbo Pascal (SYSTEM, CRT, DOS, PRINTER, GRAPH, OVERLAY, TURBO3 e GRAPH3). Cinque di queste si ritrovano anche nel Quick Pascal (mancano OVERLAY, TURBO3 e GRAPH3), accanto alle due per la grafica stile-Microsoft (MSGRAPH e MSGRUTIL) e ad altre quattro in sorgente (MOUSE, BIGHEAP, MENU e TURTLE).

MSGRAPH e MSGRUTIL consentono di scrivere programmi grafici in modo indipendente dall'hardware in modo analogo a quanto può farsi con la BGI, con qualcosa in più per quanto riguarda i sistemi di coordinate (molto flessibili), con qualcosa in meno per *la business graphics* (niente grafici a barre o a torta). MOUSE, MENU e TURTLE offrono quello che è facile indovinare: gestione del mouse e di menu a tendina, una *turtlegraphics* non compatibile con quella offerta dalla GRAPH3 del Turbo Pascal. BIGHEAP, infine, contiene implementazioni di *GetMem* e *FreeMem* che consentono di allocare e deallocare blocchi di memoria più ampi di 65520 byte. Un corredo più che accettabile, ma limitato alla programmazione tradizionale.

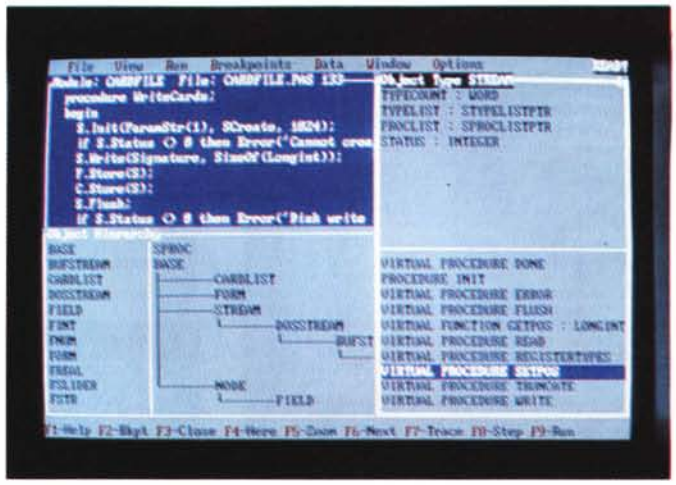
La Borland propone invece il sorgente di una unit OBJECTS che rende concreta la possibilità di programmare object oriented. Il Pascal infatti non consente di creare file di oggetti. Meglio: il Quick Pascal non si oppone alla dichiarazione di un file simile, ma se si prova a scrivervi un oggetto si scrivono in realtà quattro byte (un puntatore!), e se si prova a rileggere e usare quei byte si incorre in un errore di esecuzione. Il Turbo Pascal avverte subito, sin dalla compilazione, che certe cose non si possono fare, ma soprattutto ci dà una soluzione: la unit OBJECTS offre infatti, accanto ad una classe «lista generica», anche una classe *stream* (anch'essa ispirata dal C++): i suoi metodi rendono possibile aprire, chiudere, scrivere e leggere file di oggetti. Tre ulteriore unit CARDS, FORMS e SLIDERS e un programma CARDFILE ne offrono un suggestivo esempio.

Ambiente di sviluppo

Conosciamo bene l'ambiente integrato del Turbo Pascal e altri ambienti Quick. Il Quick Pascal offre qualcosa in più: la possibilità di aprire diverse finestre di editing, sia su più sorgenti che su zone diverse di uno stesso file (in quest'ultimo caso è però necessario creare prima una finestra vuota con Alt-File-New). Ogni finestra può essere «zoomata» e poi riportata alle sue dimensioni originali con Ctrl-F10. Si può anche lanciare il compilatore indicando fino a nove file sulla riga comando: per ognuno viene aperta una distinta finestra, e ci si può spostare rapidamente dall'uno all'altro premendo Alt e un tasto da 1 a 9. Si può scegliere tra finestre «a cascata» (una sopra l'altra) o accostate (ad ognuna una distinta zona dello schermo; soluzione efficace soprattutto se ci si limita a due o tre finestre). Il Turbo Pascal offre solo la cosiddetta «pick list» (già discussa quando provammo la versione 4.0): non altrettanto comoda e un po' più lenta, visto che per passare da un file all'altro bisogna accedere al disco.

Rimane invece alla Borland il primato della velocità di compilazione, in quanto il Quick non è in grado di compilare in memoria: anche un programma «vuoto» (un **begin** subito seguito da **end**) richiede qualche secondo; su un clone AT a 10 MHz con disco da 40 millisecondi occorrono circa 15 secondi per 1500 linee di sorgente, solo un terzo con il Turbo Pascal. La compilazione separata non fa invece registrare differenze significative: a volte un po' più veloce l'uno, a volte l'altro; si può solo rilevare che spesso il file eseguibile prodotto dal Quick Pascal è più voluminoso (il massimo l'abbiamo registrato compilando il BGIDEMO della Borland: 89488 byte contro i 41920 generati dal Turbo Pascal; forse in questo caso la colpa è tutta della unit GRAPH).

Anche le capacità del debugging integrato sono all'incirca equivalenti, pur se con qualche differenza di impostazione. Molto dipende dal gusto personale: può ad esempio risultare comoda la possibilità di scrivere direttamente nella finestra di debug i nomi delle variabili da tenere sotto controllo, come consente il Quick Pascal, può risultare più chiara e più agile la meccanica della modifica dei valori di queste variabili nel Turbo Pascal. Due note tuttavia si impongono. Dopo aver visto il debugger integrato del QuickC 2.0, quello del QP sembra costretto a recitare il ruolo del parente povero: non si può aprire una finestra dedicata alle variabili locali, non si può aprire una finestra sui registri del microprocessore, come invece sarebbe tanto comodo per le istruzioni **inline**. Il Quick Pascal non può poi usare un debugger separato (neppure il CodeView), ed è chiaro il vantaggio che il potente Turbo Debugger conferisce al compilatore della Borland, tanto più ora che è stato



La nuova versione del Turbo Debugger è in grado di visualizzare l'intera gerarchia delle classi di un programma OOP e di ispezionare i dettagli di ognuna.

arricchito di nuove opzioni per visualizzare la gerarchia delle classi di un programma e per sbirciare dentro gli oggetti.

Prestazioni

Due parole infine sulla qualità del codice prodotto. Abbiamo condotto numerose prove, ben intenzionati a pubblicare i risultati di un vero benchmark, rielaborazione di quello approntato da Wirth per i compilatori Modula-2. Abbiamo però rinunciato, in quanto pochi centesimi di secondo non significano proprio nulla. Sembra che il Quick sia un po' più veloce quando esegue le quattro operazioni sugli interi o quando assegna valori agli elementi di una matrice; sembra che il Turbo sia più veloce nelle quattro operazioni sui *single* o nelle assegnazioni agli elementi di un array monodimensionale. Ma non serve a molto prodursi in tabelle con tempi del tipo 8,19 secondi contro 7,97 o 7,85 secondi contro 8,34 quando un'operazione viene ripetuta per diecimila volte. Di analogo ordine di grandezza è il miglioramento delle prestazioni che il Quick Pascal offre a chi ha almeno un 80286: attivando una direttiva G si approfitta ove possibile delle più efficienti istruzioni di quel microprocessore. Abbiamo anche controllato il codice oggetto, per riscontrare che il QP ripropone le stesse ottimizzazioni del concorrente.

Le uniche differenze significative si osservano nell'aritmetica sui *real* e nelle funzioni trascendenti (su *single*, *real* e *double*; non abbiamo provato con *extended* e *comp* in considerazione della diversa precisione offerta dai due compilatori). Quanto ai *real*, tutto dipende da come si imposta la direttiva N del Turbo Pascal: se si attiva l'emulazione del coprocessore le operazioni aritmetiche sono un venti per cento più lente che nel QP, altrimenti sono più veloci del 50%; quanto a seni coseni e logaritmi, il Quick Pascal è sempre nettamente più lento: per calcolare ventimila seni o coseni richiede un minuto e venti secondi, un minuto e quaranta per ventimila logaritmi; il Turbo Pascal si ferma rispettivamente a 37 e 49 secondi (sempre su un AT a 10 MHz).

Parlando di efficienza del codice bisogna poi fare anche tutto un altro discorso: quanto «costa» la programmazione object oriented? Abbiamo accennato nel numero scorso ai suoi innegabili vantaggi, ma è lecito aspettarsi che i metodi virtuali, in quanto devono trovare al momento della esecuzione il codice da eseguire, comportino qualche perdita di velocità. Così sembrerebbe, e così ha sostenuto perfino Byte. In realtà non basta fermarsi a misurare il mezzo secondo o i due secondi circa in più (tanta è la differenza tra l'esecuzione di TRECENTOSESSANTAMILA chiamate di procedura e altrettante chiamate di metodi virtuali, con nessuno o con quattro parametri); occorre considerare quanto la OOP ci offre di più e di meglio. In concreto, abbiamo provato a tradurre in programmazione tradizionale l'esempio di «lista di qualsiasi cosa» proposto a luglio: ne è venuto fuori un codice zeppo di if e case, e soprattutto ben difficilmente modificabile se non riscrivendo tutto quanto; altro che aggiungere un tipo senza dover nemmeno ricompilare le unit degli oggetti-base! Come se non bastasse, *non abbiamo riscontrato nessuna differenza nei tempi di esecuzione*! In altri termini, cercare di realizzare in modo tradizionale quel che si può fare con la OOP è una vera pena per il programmatore (Jobs non ha scelto l'Objective-C per il suo NeXT solo per tener dietro ad una moda), e se ci si prova si deve comunque «appesantire» il codice. Meglio allora seguire la via più facile, senza temere perdite di efficienza che in realtà non vi sono.

Conclusioni

Due prodotti interessanti, in quanto ambedue consentono di appropriarsi delle più avanzate tecniche di programmazione senza traumi, senza dover imparare nuovi linguaggi.

La Borland non ha fatto altro che continuare per la sua strada. Sostenendo che C e Pascal devono migliorarsi l'uno prendendo esempio dall'altro, dopo aver messo un po' di C fin nella prima versione del suo compilatore, propone ora ai suoi utenti una evoluzione analoga a quella che il C++ propone ai fedeli del C. E lo fa alla sua maniera: con una implementazione originale ma coerente, potente ma efficiente, supportata da ottimi strumenti di sviluppo e di debugging, ampiamente documentata e arricchita da esempi che vanno ben al di là del semplice «demo». Il tutto senza aumentare il prezzo, offrendo anzi interessanti possibilità di upgrade; segnaliamo ad esempio che chi passasse dalla versione 5.0 normale alla 5.5 Professional pagherebbe in pratica solo l'Assembler/Debugger, beneficiando così di un upgrade gratuito. Può magari restare l'indecisione se optare per il 5.5 normale o Professional (comprendente anche il Turbo Assembler/Debugger 1.5), ma va riferito che, discutendone con la EDIA, è emerso che molti hanno acquistato prima il 5.0 normale e poi in un secondo tempo il Debugger, con il risultato di spendere inutilmente centomila lire in più.

Il Quick Pascal è un compilatore ampiamente aderente allo standard di mercato e in grado di produrre codice di buona qualità. Il suo limite principale è forse quello di essere «troppo poco Microsoft» e «troppo poco Borland». Troppo poco Microsoft per via della rinuncia alla emulazione software del coprocessore numerico, della mancanza di un debugger separato, della incompleta documentazione, delle piccole incoerenze nell'help in linea e delle piccole oscurità di alcuni messaggi d'errore; non ha la stessa qualità del QuickC 2.0 — che offre un eccellente debugger integrato e buone possibilità di *business graphics* — e nulla svela della unit GRAPH, con la quale si avrebbe accesso alla ricca libreria grafica della BGI. Troppo poco Borland perché tra le estensioni OOP dei due compilatori non c'è paragone: tanto scarne ed essenziali quelle del Quick Pascal, quanto articolate ed efficaci quelle del Turbo Pascal, anche meglio documentate e supportate. C'è da scommettere che prima o poi vedremo un Quick Pascal 2.0 di ben altra fattura. Già la versione 1.0 ha comunque qualcosa da dire, grazie al miglior editor e al prezzo più conveniente, e potrà sicuramente giovarsi della possibilità di girare sotto Windows o di farsi usare con il mouse.

UN RAGGIO DI LUCE PER COMPOSIZIONI ARMONICHE.

ADVERTEAM

STAMPANTI LASER La tecnica di stampa basata sul raggio laser è in continua evoluzione e risulta sempre più vincente per chi richiede: alta qualità di stampa, assoluta silenziosità, grande capacità grafica, diversificazione delle fonti di stampa e velocità di produzione. In questo settore tecnologico la Mannesmann Tally propone diversi modelli orientati al trattamento testi e al DTP.

Non accontentatevi di una stampante qualunque, scegliete:

MT 910 e 910 UPS ■ Velocità 10 pagine al minuto ■ Doppio cassetto alimentazione fogli ■ Stampa su lucidi, buste, etichette ■ 512 K standard ■ Espansione 1,5 Mbyte opzionale ■ Volume di stampa 5.000 pagine mese ■ Versione Desk Top Publishing con linguaggi PDL e DDL ■ Risoluzione 300 dpi

MT 905 ■ Velocità 6 pagine al minuto ■ Volume di stampa 3.000 pagine mese ■ Risoluzione 300 dpi ■ Interfaccia parallela, seriale e RS 422 ■ 512 Kbyte standard

MANNESMANN
TALLY
Stampanti in assoluto

MANNESMANN TALLY srl - 20094 Corsico (MI) - Via Borsini, 6 - Tel. (02) 4502850/855/860/865/870 - Telex 311371 Tally I - Fax (02) 4500934 ■ 00144 Roma - Via M. Peroglio, 15 - Tel. (06) 5984723/5984406 - Fax (06) 5880914 ■ 10099 San Mauro (TO) - Via Casale, 308 - Tel. (011) 8225171 ■ 40121 Bologna - Via Amendola, 8 - Tel. (051) 523380 ■ 35133 Padova - Via Pontevigodarzere, 250 - Tel. (049) 8870038 ■ 50127 Firenze - Via Caduti di Cefalonia, 52 - Tel. (055) 433994