Borland
Turbo Debugger 1.0

di Sergio Polini

Come racconta Dennis Ritchie, Unix non fu progettato per rispettare particolari specifiche, ma solo per fornire a Ken Thompson, e poi allo stesso Ritchie, un ambiente piacevole per la scrittura e l'uso di software. E fu un successo. Come racconta sempre Ritchie, il C venne creato per consentire una più comoda scrittura di Unix; alcune sue caratteristiche, come la stretta parentela tra array e puntatori e in genere la notevole libertà nella manipolazione dei «tipi», vennero introdotte soprattutto per rendere più agevole la traduzione del software già scritto in B, linguaggio senza «tipi»; poco importava che in quel periodo (1972) i sostenitori dell'Algol 68 o del Pascal teorizzassero la necessità di uno «strong type checking». E fu un successo. Unix e C sono come sono perché i loro autori hanno preferito il criterio della «comodità d'uso» a quello della aderenza a qualche canone teorico o a qualche standard di mercato. E sono diventati pietre miliari nella storia dell'informatica.

Forse la Borland non è ancora nella storia, ma a Scotts Valley si regolano nello stesso modo. Il Turbo Pascal nacque perché i suoi autori trovavano scomode altre implementazioni del Pascal, perché volevano un compilatore che potessero loro per primi usare speditamente: centinaia di migliaia di utenti hanno mostrato di condividere il loro punto di vista. Chiunque programmi ha costante bisogno di una tabella ASCII e di una calcolatrice in binario ed esadecimale; sul tavolo di un programmatore (compreso quello su cui questa prova è stata condotta) è ben difficile che ci sia

Turbo Debugger 1.0

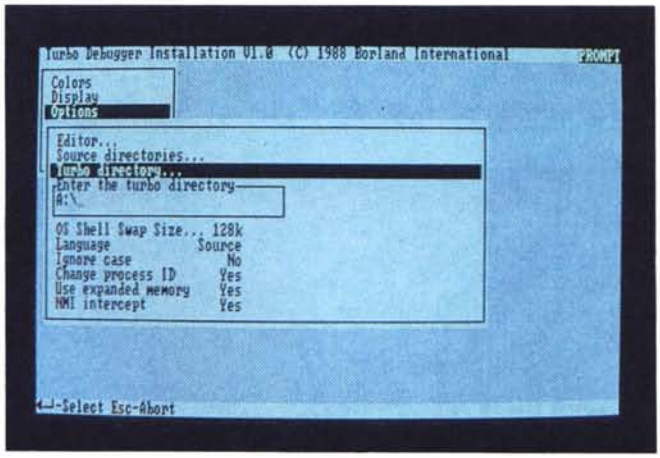
Produttore
Borland International
1800 Green Hills Road
P.O. Box 660001
Scotts Valley, CA 95066-0001
Distributore
Edia Borland Srl
Via Cavalcanti, 5 - 20127 Milano
Telefono: 02/2610102
Prezzo *(IVA 9% esclusa):*
Turbo Assembler/Debugger 1.0 L. 249.000

lo spazio per penne matite e pezzi di carta; cosa c'è di più comodo che poter richiamare la tabella, la calcolatrice e un blocco d'appunti elettronico con la semplice pressione di un tasto? Quando il SideKick, creato in origine per un uso interno, venne proposto al mercato, centinaia di migliaia di utenti aderirono convinti. Chi vende software deve anche preparare manuali; da qualche tempo la Borland scrive i suoi manuali con Sprint, il word processor esaminato a dicembre. Anche qui si tratta di un prodotto realizzato in primo luogo per un uso interno, tanto che, pur sapendosi della esistenza di Sprint, non era sicuro se sarebbe stato messo in vendita oppure no. Quando finalmente è stato proposto al mercato, si è potuto constatare che si tratta di un wp estremamente flessibile, capace di offrire sia la familiare immediatezza d'uso del WordStar che i sofisticati controlli di stampa di Word, riuscendo a rendere comoda per l'utente sia una scrittura «di getto» che la preparazione di testi lunghi e complessi.

Sembra di poterne ricavare una semplice massima: quando un programma viene proposto da chi ne aveva bisogno in primo luogo per sé, da chi voleva per sé uno strumento al tempo stesso potente e facile da usare, anche l'utente non può che trarne la massima soddisfazione. Chi programma ha bisogno di un buon debugger ancor più di una calcolatrice esadecimale, e c'è da scommettere che la vera «indagine di mercato» per il Turbo Debugger è stata condotta nell'ambito dello staff tecnico della stessa Borland. Un altro esempio della validità di quella massima?

Quando abbiamo per la prima volta dato il comando TD al prompt del DOS eravamo in realtà un po' sospettosi: abbiamo illustrato il mese scorso, in occasione della prova del Turbo Assembler, i motivi che rendevano urgente per la Borland la commercializzazione di un debugger simbolico. Il Turbo Debugger poteva anche essere solo il risultato di una frettolosa adesione ai nuovi standard di mercato, poteva anche essere solo una poco convinta imitazione del CodeView della Microsoft. La primissima impressione è stata, lo confessiamo, negativa: troppo diverso appariva il debugger Borland dal formidabile concorrente. Ma dopo poche ore si è manifestata in tutta la sua evidenza la incredibile qualità dei prodotti di Scotts Valley: il Turbo Debugger è «diverso» perché è tutt'altro che una imitazione; si propone invece con estrema autorevolezza come il nuovo standard. Così come il Turbo Pascal ha dato origine a tutta una generazione di compilatori interattivi, così come il SideKick è stato il punto di riferimento di tanti programmi «residenti», il Turbo Debugger ridefinisce il concetto stesso di debugger simbolico: ogni futuro prodotto analogo non potrà non essere confrontato con il modello proposto dalla Borland; in ogni potenziale concorrente si cercheranno caratteristiche quali la possibilità di modificare il sorgente del programma che si sta esaminando o i suoi file di dati, di «zoommare» su strutture di dati complesse, di impostare breakpoint «attivi». Per non parlare del debugging «remoto» o del debugging «virtuale» su sistemi con 80386.

Ma forse conviene procedere con ordine.



Il programma TDINST consente di configurare il Turbo Debugger secondo i propri gusti. Se si intende installare il prodotto su un sistema dotato di due floppy da 360K, è necessario portare sul «disco di lavoro» (normalmente nel drive B:) il file TD.OVL e lasciare nell'altro TD.EXE e TDHELP.TDH. Con l'opzione Turbo Directory *è possibile dire al Debugger dove trovare i vari file di cui ha bisogno (in particolare il file con gli help).*

Scritto in Turbo C, il Turbo Debugger richiede 384K di Ram, un DOS 2.0 o successivo e programmi compilati con il Turbo C 2.0, il Turbo Pascal 5.0 o il Turbo Assembler; si trova a suo agio soprattutto su un disco rigido, ma può essere usato anche con due floppy.

La procedura di installazione su due floppy da 360K non è macchinosa; riteniamo tuttavia utile esporla in dettaglio, dal momento che non ci sembra illustrata nel modo migliore né sui manuali né nel file README. Si assegna in primo luogo alla variabile PATH dell'environment la directory in cui risiederà il Debugger («A:\»); si copiano poi sul dischetto da tenere nel drive A: i file TD.EXE e TDHELP.TDH (per complessivi 280K), sul dischetto da tenere in B: i file TD.OVL e TDINST.EXE. Fatto partire TDINST, si sceglie prima l'opzione *Options* dal menu principale, poi quella *Turbo Directory* dal menu successivo, indicando anche qui «A:\». Con un ESC si ritorna al menu principale, dove si sceglie *Save*: si può a questo punto o creare un file di configurazione (TDCONFIG.TD è il nome di default) o modificare direttamente il programma TD.EXE. Riteniamo preferibile creare un file di configurazione, in quanto è possibile crearne anche più d'uno, da tenere in diversi dischetti o directory, o anche, ognuno con il suo nome, nello stesso posto: il Turbo Debugger può infatti caricare in qualsiasi momento un file di configurazione, in modo da adattarsi rapidamente alle più disparate esigenze. Quasi inutile dire che durante l'installazione si possono scegliere molte altre cose: dall'ampiezza dell'intervallo di tabulazione al tipo di monitor, dai colori delle diverse finestre al formato per la rappresentazione dei valori interi (decimale, esadecimale o entrambi), dal tasto con cui interrompere l'esecuzione del programma «debuggato» (Ctrl-Break per default: prezioso per i loop infiniti...) all'ampiezza massima della *watch window* (da 1 a 20 righe), fino al pathname del vostro editor preferito, con il quale potrete correggere immediatamente gli errori dei sorgenti senza uscire dal Debugger.

Se alla fine si cancella TDINST.EXE (78K), rimarranno circa 200K per i programmi da esaminare e i relativi sorgenti.

Ancora più semplice l'installazione su disco rigido: si copiano i tre dischetti del Debugger in una apposita subdirectory, si aggiunge il nome di questa al PATH, si configura eventualmente il tutto con TDINST.

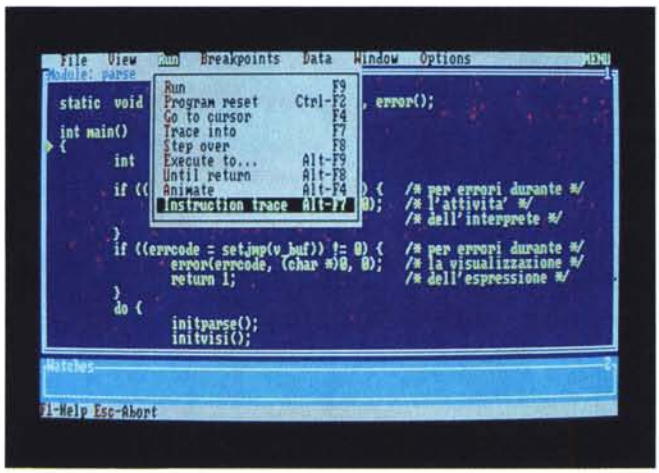
Primo impatto

Appena partito, il Turbo Debugger presenta una schermata analoga a quella del Turbo Pascal 5.0: lì avevamo due finestre *Edit* e *Watch*, qui una finestra *Module*, dove possiamo esaminare il sorgente e seguire passo passo l'esecuzione, e una finestra *Watches*, dove potremo tenere sotto costante controllo il valore di variabili o espressioni, proprio come abbiamo già visto il mese scorso per le ultime versioni del Turbo C e del Turbo Pascal, con i loro debugger integrati. Basta tuttavia dare un'occhiata al menu *Run* per avere già una prima parziale idea delle maggiori potenzialità del debugger separato rispetto a quello integrato. Alcuni comandi sono uguali: *Run* (F9) esegue il programma fino al primo breakpoint o fino alla fine (o fino ad un Ctrl-Break), *Program reset* (Ctrl-F2) ci riporta ad una situazione identica a quella che si aveva subito dopo aver fatto partire il Debugger, *Go to cursor* (F4) esegue il programma fino alla riga su cui si trova il cursore, *Trace* (F7) e *Step* (F8) eseguono il codice corrispondente alla riga su cui si trova il cursore, anche qui con la differenza che *Trace* ci porta fin dentro le funzioni o procedure che vengono man mano chiamate, mentre *Step* salta subito all'istruzione immediatamente successiva alla chiamata.

Poi ci sono le novità, alcune delle quali sono semplici variazioni su temi già noti. Con *Execute to* (Alt-F9), ad esempio, si può eseguire velocemente il programma caricato fino ad un punto specificato in una apposita «dialog box»; l'indirizzo al quale si deve arrivare può essere indicato in vari modi: il nome di una funzione, una espressione il cui valore sia un indirizzo nel code segment, il nome di un file sorgente seguito da un numero di riga. Con *Animate* (Alt-F4), vengono eseguite l'una dopo l'altra le istruzioni del programma come se si dessero ripetuti comandi *Trace*, fino alla pressione di un qualsiasi tasto.

Until return è invece una piccola ma utilissima invenzione Borland. Se usate il comando *Trace*, «saltate dentro» ogni funzione o procedura che venga chiamata dalla istruzione su cui è il cursore; potrebbe essere proprio questo quello che volete, ma può anche capitare che in realtà volevate usare *Step*, oppure che vi accorgete che non vi interessano i dettagli della procedura in cui siete «saltati». Nessun problema: con Alt-F8 potete fare un altro salto in avanti, questa volta però alla istruzione successiva alla chiamata della procedura, ritornando così subito dove eravate.

Con *Instruction trace*, infine, si può cominciare ad avere una idea più precisa della potenza del Turbo Debugger. Si apre una finestra *CPU* articolata in cinque pannelli: le istruzioni assembler generate dal compilatore, lo stato dei registri del microprocessore, il valore corrente dei flag, il «dump» di un'area di memoria (per default l'inizio del data segment corrente), il contenuto dello stack. All'inizio il cursore si trova nel primo pannello, nel quale si possono dare gli stessi comandi *Trace*, *Step*, ecc. che abbiamo appena visto, ma è possibile spostarsi dall'uno all'altro con il tasto di tabulazione oppure premendo Shift e uno dei tasti-freccia. Soprattutto è possibile cambiare le istruzioni assembler, alterare il contenuto dei registri o del data segment o dello stack... semplicemente scrivendovi sopra! È possibile cambiare il valore dei flag semplicemente portandosi lì con il cursore e premendo Enter! E non è tutto!



Il Turbo Debugger, appena partito, presenta una schermata simile a quella del Turbo Pascal 5.0: una finestra «Module» al posto di quella «Edit», e subito sotto una finestra «Watches». Basta però premere Alt-R (o F10 e poi R) per avere una prima parziale idea delle maggiori potenzialità del debugger separato rispetto a quello integrato: le opzioni con Alt del menu «Run» sono nuove.

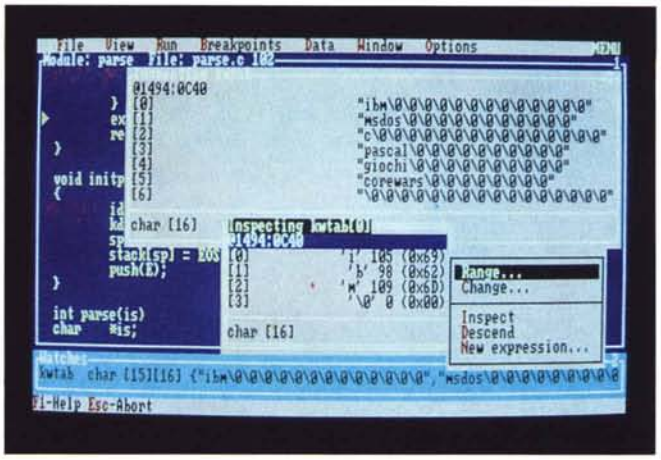
Tanti piccoli menu...

Le opzioni del menu principale (la prima riga del video) possono essere attivate in vari modi: premendo Alt e l'iniziale dell'opzione desiderata, premendo F10 e poi l'iniziale, oppure premendo F10 e posizionandosi con i tasti-freccia. Ad ogni opzione corrisponde un ricco menu, come abbiamo appena visto per *Run*: un totale di sette menu che, quasi inutile dirlo, sono sempre gli stessi, qualsiasi cosa si stia facendo.

«Quasi inutile» perché premendo Alt-F10 succede qualcosa di ben diverso: ci sono, come vedremo, undici diversi tipi di finestre, alcune delle quali (come la *CPU*) divise in più pannelli; una moltitudine di ambienti ad ognuno dei quali è associato un suo menu «locale».

Prendiamo le finestre che abbiamo già visto. Si può scorrere il sorgente mostrato in *Module* con gli stessi comandi degli editor del Turbo C e del Turbo Pascal, con qualche limitazione: alcune combinazioni di tasti svolgono infatti una funzione diversa. Portandosi con il cursore su una variabile o su una espressione (in questo caso occorre «marcarla» premendo il tasto INS ad un suo estremo e facendo poi scorrere il cursore fino all'altro), basta un Ctrl-W per aggiungerla alla finestra *Watches*. Più in generale hanno un significato nuovo i comandi Ctrl-<lettera>, dove «lettera» è l'iniziale di una delle opzioni del menu locale a *Module*. Queste consentono di passare da un file sorgente ad un altro (*Module, File, Previous*), di cercare una stringa (*Search, Next*), di andare ad un dato indirizzo (*Goto*; l'indirizzo va immesso come sopra abbiamo visto per il comando *Execute to*), di tornare alla riga corrente del sorgente, quella che verrebbe eseguita con *Trace* o *Step* (*Origin*). C'è poi un'opzione che carica l'editor eventualmente scelto durante l'installazione (*Edit*), con il quale è possibile correggere subito il sorgente nel quale si sia scoperto un errore. E c'è anche una opzione *Inspect*, di cui parleremo tra un attimo.

Non si creda che l'utente è costretto a memorizzare i nomi di tutti i comandi: normalmente sull'ultima riga dello schermo compaiono i comandi che è possibile attivare premendo i tasti funzione; se però si tengono premuti per



Premendo Alt-F7 si apre una finestra in cinque pannelli: le istruzioni assembler generate dal compilatore, i registri del microprocessore, i flag, il «dump» di un'area di memoria (per default l'inizio del data segment), lo stack. È possibile alterare direttamente il contenuto di ognuno dei pannelli, semplicemente iniziando a scrivervi sopra: si apre immediatamente una «dialog box» il cui contenuto andrà poi a rimpiazzare i valori su cui è il cursore. Si può notare (nel pannello che mostra l'inizio data segment) che il file parse.c è stato compilato con un compilatore Microsoft (il C 5.1): una apposita utility modifica infatti i file .EXE preparati per il CodeView in modo che sia possibile esaminarli con il Turbo Debugger.

Il comando Inspect *consente di esaminare strutture di dati troppo complesse per essere rappresentate in una riga della finestra* Watches. *Si apre una nuova finestra, alla quale è associato un menu «locale» (richiamabile con Alt-F10) che comprende diversi comandi: tra questi* Inspect *e* Descend *permettono di esaminare con lo stesso dettaglio strutture che siano a loro volta comprese in quella di partenza (ad esempio: uno degli array di un array di array, strutture «puntate» da campi puntatori di un'altra, e così via),* Descend *nella stessa finestra,* Inspect *in un'altra appositamente aperta.*

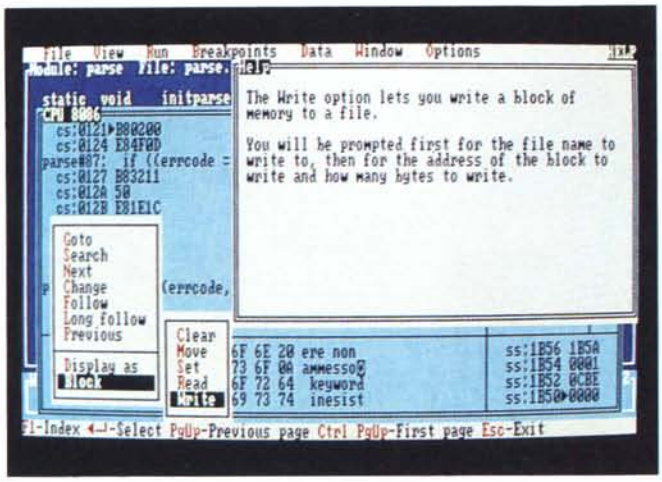
un secondo i tasti Ctrl o Alt, su quella riga compare un sintetico help sulle varie combinazioni disponibili.

Se si preme Alt-F10 quando si è nella finestra *Watches*, appare un menu locale che consente di aggiungere variabili o espressioni a quelle eventualmente già sotto osservazione (*Watch*), di cancellarne una o tutte (*Remove* e *Delete all*), di cambiare il valore delle variabili (*Change*) o di modificare direttamente (senza cioè cancellare e poi riscrivere) il nome di una variabile o la sintassi di una espressione (*Edit*). Naturalmente tutti i comandi possono essere dati anche senza passare per il menu con Alt-F10: basta premere Ctrl e la lettera iniziale dei vari comandi. Naturalmente anche qui c'è un comando *Inspect*.

Si tratta di una delle innovazioni più significative rispetto al CodeView (nella versione 2.1 di questo c'è in realtà un comando *Graphic Display*, da dare con due punti interrogativi, che fa qualcosa di analogo, ma in modo nettamente più limitato). Se nei programmi ci fossero solo variabili non strutturate la finestra *Watches* sarebbe più che sufficiente. Per fortuna i nostri linguaggi di programmazione ci consentono di usare array, strutture, record, liste concatenate, ecc. Se un array ha 100 elementi non si può esaminarlo tutto su una riga di *Watches*, né si possono tenere costantemente sotto osservazione tutti i suoi elementi su righe diverse. Nei debugger integrati del Turbo C 2.0 e del Turbo Pascal 5.0 si è risolto il problema con uno scrolling orizzontale delle righe, ma non è così facile seguire i nodi di una lista concatenata. Nel Turbo Debugger si può fare di meglio: il comando *Inspect* apre una nuova piccola finestra grazie alla quale ci si può «affacciare» su qualsiasi variabile strutturata. Ogni elemento del nostro array o della nostra struct (o record) ci viene mostrato su una riga. Se la piccola finestra non basta, con il tasto F5 si può «zoommare» in modo da farle occupare tutto lo schermo; se così fosse troppo, premendo il tasto Scroll-Lock si possono cambiare con i tasti-freccia come si desidera le dimensioni della finestra di *Inspect* (come del resto di ogni altra finestra del Turbo Debugger) e si può anche spostarla nel punto che più ci è comodo dello schermo. Se poi su una o più righe viene rappresentato un elemento a sua volta strutturato della nostra variabile, basta andare lì con il cursore e dare un nuovo comando di *Inspect*, e così via fino ai tipi di dati più elementari. Se non si vuole riempire lo schermo di tante finestrelle, il «piccolo menu» di *Inspect* offre anche un comando *Descend*, che opera nello stesso modo, rimanendo però sempre nella stessa finestra. Questo vuol dire che posso aprire una finestra sul nodo-radice di una lista, andare con il cursore sul campo «puntatore al nodo successivo», e scorrermi nella stessa finestra tutta la lista.

Ugualmente ricchi i menu locali ai diversi pannelli della finestra *CPU*. Nel pannello che mostra la codifica in assembler si ritrovano le stesse opzioni *Goto, Origin* e *Search* che abbiamo incontrato in *Module*, con significato analogo (con *Search* si possono cercare sequenze di byte o anche istruzioni espresse in forma simbolica, ad esempio «PUSH AX»), accanto ad altre che permettono di scegliere il tipo di visualizzazione (le righe del sorgente seguite dalla loro codifica in assembler, o solo assembler: *Mixed*), di aprire una finestra *Module* per guardare al sorgere (*View source*), di dare un'occhiata al codice che verrebbe eseguito se si desse un comando di *Trace* su una CALL o un JMP e poi tornare indietro (*Follow, Caller, Previous*). Con altri comandi si può modificare l'istruzione su cui è il cursore scrivendone un'altra in forma simbolica (*Assemble*), far proseguire il programma da una istruzione a nostra scelta (*New cs:ip*), leggere o scrivere un byte o una word su una porta (*I/O*).

Opzioni come *Goto, Origin, Follow* e *Previous*, con un *Change* al posto di *Assemble*, si ritrovano anche nei menu locali ai pannelli aperti su una generica area di memoria o sullo stack; per il pannello di «dump» c'è comunque qualcosa in più: una opzione *Long follow* permette ad esempio di interpretare i byte su cui è il cursore come un «far pointer» (coppia offset/segmento) e di saltare all'area di memoria da questo «puntata», mentre con *Display as* si può scegliere il formato della rappresentazione tra tutti quelli disponibili sui



L'opzione Block del menu locale al pannello di «dump» permette di cancellare o muovere un blocco di byte, di riempirlo con un valore a scelta o con il contenuto di un file, di scriverlo su un file. La figura illustra quest'ultima opzione e la relativa schermata di help. Notate che è possibile spostare la finestra di help dove si vuole, anche se, a differenza di tutte le altre finestre del Turbo Debugger, non è possibile cambiarne le dimensioni.

compilatori Borland (byte, word, long, comp, float, real, double, extended), e con *Block* si può cancellare o muovere un blocco di byte, riempirlo con un valore a scelta o con il contenuto di un file, scriverlo su un file.

I menu locali ai pannelli aperti sui registri e sui flag consentono infine di modificarne i valori.

... per tante finestre

L'opzione *View* del menu principale apre un altro ricco menu, dal quale si possono aprire varie finestre. Ne conosciamo già alcune: *Module, Watches* e *CPU*, ma anche *Dump* e *Register*: è possibile infatti aprire una finestra limitata ad uno solo di questi due pannelli della *CPU*. Non conosciamo invece *Stack* che, a differenza del corrispondente pannello della finestra *CPU*, mostra l'elenco delle funzioni o procedure «attive», quelle cioè attraverso le quali siamo arrivati mediante successive chiamate al punto in cui siamo, ma alle quali non siamo ancora ritornati.

Il «piccolo menu» di *Stack* ha solo due opzioni: con *Inspect* si apre una nuova finestra *Module* contenente il sorgente della funzione su cui è il cursore, con *Locals* si apre una finestra di un nuovo tipo, *Variables*, che ne mostra i simboli locali con i relativi valori (si tratta di una caratteristica molto utile soprattutto nel caso di ripetute chiamate di una funzione ricorsiva). Anche la nuova finestra ha un suo «piccolo menu», con le ormai familiari opzioni *Inspect* e *Change*.

Si può aprire una finestra *Variables* anche dal menu *View*: in questo caso si possono osservare, ed eventualmente cambiare, i valori di tutti i simboli globali del programma, oltre a quelli locali al file e alla funzione su cui si è posizionati.

Vi è poi una finestra *Numeric processor*, che mostra il contenuto dei registri dell'80×87, nel caso che ce ne sia effettivamente uno installato, ma anche se al programma in esecuzione è stata linkata la libreria di funzioni di emulazione software del Turbo C 2.0 o del Turbo Pascal 5.0. Naturalmente è possibile cambiare i valori sia dei registri che dei bit di controllo e di stato.

Con la finestra *File* si può esaminare il contenuto di qualsiasi file, sia in ASCII che in esadecimale: il solito «piccolo menu» permette di scegliere il modo di rappresentazione che più ci è comodo (*Display as*), che condiziona anche il funzionamento delle altre opzioni; *Goto, Search* e *Next* funzionano come in *Module* se si è scelto il modo ASCII, come in *Dump* se si è scelto il modo esadecimale; *Edit* chiama l'editor eventualmente scelto in fase di installazione se si è in modo ASCII, consente di modificare direttamente i singoli byte se si è in modo esadecimale.

Si possono poi esaminare nella omonima finestra i breakpoint eventualmente impostati, o si può aprire la *Log window*. In questa vengono registrati automaticamente tutti i breakpoint attraverso cui il programma in esame è passato durante l'esecuzione, e, mediante una opzione del menu principale (*Window/Dump pane to log*) vi si possono aggiungere in qualsiasi momento i contenuti di un'altra finestra; grazie al «piccolo menu» di turno si può poi anche scegliere di scrivere contemporaneamente, oltre che nella *Log window*, anche in un *Log file*, e di aggiungervi propri commenti. Sarà così facile ricostruire quello che è successo durante una tormentata sessione di debugging.

Tra tante finestre ci si potrebbe anche perdere. Per fortuna, oltre ad aprirle, possiamo anche chiuderle (con F3) o, se cambiamo idea, riaprire quella appena chiusa (con Alt-F6); ogni finestra è numerata, e possiamo passare dall'una all'altra, secondo l'ordine della loro numerazione, con il tasto F6, oppure saltare direttamente a quella che ci interessa premendo contemporaneamente Alt e il tasto con il numero; se non ricordiamo questo, con Alt-0 si apre una finestrella con un elenco completo: ci si posiziona con il cursore su quella desiderata e si preme Enter. Se non ci piacciono le dimensioni o la posizione di una finestra, premiamo Scroll-Lock e usiamo i tasti-freccia. Se infine non ricordiamo tutte queste possibilità, non dobbiamo fare altro che scegliere l'opzione *Window* del menu principale. Se non ci è chiaro il senso di una qualsiasi opzione di un qualsiasi menu, basta premere F1 per avere una pertinente schermata di help. Non si potrebbe chiedere di più.

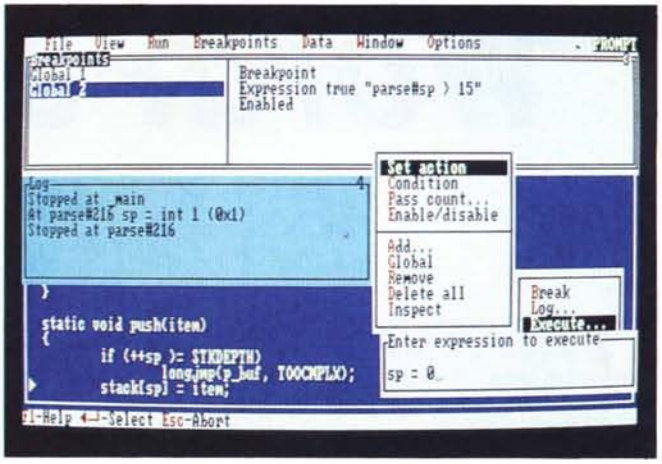
Breakpoints

L'omonima opzione del menu principale sembra aprire un menu piuttosto povero: con *Toggle* (F2) si imposta un «normale» breakpoint alla riga su cui è il cursore, con *At...* (Alt-F2) si può impostare un «normale» breakpoint ad un indirizzo da specificare. Ovvio il significato di *Delete all*, mentre le altre due opzioni equivalgono l'una ai *Watchpoint* e l'altra ai *Tracepoint* del CodeView (un breakpoint è un punto in cui l'esecuzione del programma si arresta, un watchpoint causa l'arresto del programma quando una data espressione diventa vera, un tracepoint causa l'arresto quando cambia il valore di un'area di memoria). Ordinaria amministrazione.

In realtà la Borland ha invece ridefinito il concetto di «breakpoint». Questo viene inteso come un'azione che viene eseguita in un certo luogo se si verificano certe condizioni. Quanto al luogo, un breakpoint può essere locale o globale; nel primo caso scatta sempre quando il programma raggiunge una data istruzione, nel secondo solo se si verificano alcune condizioni: se cambia il valore di un'area di memoria (ad esempio una variabile) o se diventa vera una espressione. Quanto alle azioni, l'effetto di un breakpoint può essere non solo quello di arrestare l'esecuzione del programma, ma anche di scrivere qualcosa nella *Log window* (il valore di una variabile o di una espressione), e addirittura di eseguire una istruzione, come se questa fosse stata inclusa nel sorgente del programma e questo ricompilato!

A tutte queste possibilità si accede, stranamente, dal menu *View*: scegliendo l'ozione *Breakpoints* si apre una finestra nella quale viene mostrato lo stato dei breakpoint impostati fino a quel mo-

mento, ma alla quale è anche associato un potente «piccolo menu». È da qui che si sceglie il tipo di azione o la condizione; si possono anche temporaneamente disabilitare e poi riabilitare i breakpoint già impostati, o anche definire un *Pass count*, il numero di volte che la condizione si deve verificare prima che venga eseguita l'azione.



Un breakpoint può essere locale (scatta quando il programma arriva ad una data istruzione) o globale (quando cambia il valore di un'area di memoria o diventa vera una espressione). Quando scatta può produrre diversi effetti: interrompere l'esecuzione, scrivere qualcosa nella Log window, *o anche eseguire una istruzione, come se questa fosse stata aggiunta al programma.*

Accessori

Quanto fin qui visto non può essere descritto che come un debugger simbolico molto completo. Gli «accessori» sono all'altezza del prodotto.

In primo luogo il manuale, tanto necessario quanto ben fatto: necessario per guidare l'utente in un ambiente (anzi in una moltitudine di ambienti) di notevole potenza e complessità, chiaro ed esauriente in misura tale da non lasciare spazio a dubbi o ad ambiguità. Vi sono persino capitoli di efficace impronta didattica, che illustrano i bug più comuni per chi programmi in C, Pascal o assembler, mentre una appendice tecnica contiene preziose informazioni su aspetti come gli interrupt usati dal Debugger o l'uso della memoria espansa eventualmente disponibile.

Vi sono poi alcuni programmi di utilità: da TDCONVRT che converte da formato CodeView a formato Turbo, a TDMAP, che aggiunge ad un file EXE le informazioni contenute nel file MAP creato dal linker, in modo da rendere possibile il debugging simbolico anche di programmi compilati con prodotti né Borland né Microsoft; da TDPACK, che comprime le informazioni simboliche contenute in un file EXE, a TDSTRIP che le elimina del tutto riportando il file a dimensioni «normali», a TDUMP, che mostra la struttura di un file EXE, OBJ o LIB.

Ma ci sono anche TD386.EXE e TDH386.SYS, TDREMOTE e TDRF.

Se si dispone di una macchina con 80386 con almeno 700K di memoria estesa, i primi due file consentono di caricare il Debugger in questa, lasciando al programma da esaminare tutti i 640K di quel PC virtuale che un 80386 è in grado di gestire.

Se non si ha un 80386, e se il programma da spulciare è troppo ingombrante per dividere con il Debugger i consueti 640K, si può risolvere il problema collegando le porte seriali di due computer con un cavo «null modem»: su una macchina sarà sufficiente far partire TDREMOTE (che occupa circa 16K), sull'altra si avvia il Debugger con le opzioni «–rp1» (o–rp2» se si usa la porta COM2) e «–rs3» (per un collegamento a 115K baud; si possono usare «rs1» o «rs2» per velocità minori). Il Turbo Debugger provvederà a trasferire il file EXE da esaminare sul sistema remoto, e tutto avverrà come già visto, con la differenza che l'input da tastiera e l'output su video del programma avverranno sul sistema remoto. TDRF si può invece occupare del trasferimento tra le due macchine di altri file (ad esempio file di dati).

L'estrema utilità di questa soluzione è evidente, soprattutto in abienti di studio o di lavoro, dove non manca certamente il collega a cui chiedere in prestito per qualche tempo una macchina.

Problemi

Un paio di mesi di uso intenso del Turbo Debugger ci hanno consentito di riscontrare persino qualche piccolo difetto.

È possibile memorizzare nel file di configurazione alcune sequenze di tasti: si parte con un comando *Create macro*, si danno altri comandi, si chiude con *Stop recording*, si salva tutto in un file. L'ennesima utile caratteristica di questo prodotto, soprattutto se si deve esaminare più volte una stessa situazione, con diverse variabili sotto osservazione, diversi breakpoint di vario tipo, e così via. Se i comandi registrati in una macro agiscono anche su dimensioni e posizioni delle finestre, tuttavia, è facile non ottenere l'effetto desiderato, tanto che avevamo pensato ad un vero e proprio malfunzionamento. In realtà funziona tutto a dovere (basta leggere con attenzione il manuale e il file MANUAL.DOC e fare un po' di prove), ma forse sarebbe desiderabile una operatività un po' più agile.

È possibile esaminare con il Turbo Debugger anche programmi compilati per il CodeView. Abbiamo riscontrato che qualche volta però TDCONVRT si impunta durante la conversione (anche se è successo su un PS/2 sul quale il CodeView si rifiuta proprio di partire, sparando un misterioso «Internal error»), ed è necessario ripetere. La seconda volta va comunque sempre tutto bene. O quasi. Qualche volta il comando *Inspect* mostra uno sfasamento di un byte quando rappresenta strutture di dati particolarmente complicate, forse a causa delle diverse opinioni della Microsoft e della Borland circa l'allineamento in memoria degli elementi di una struttura; questo tuttavia non ci ha mai impedito un efficace debugging anche di programmi complessi compilati con il Microsoft C 5.1.

Tutto qui.

Conclusioni

Che dire? Gli unici difetti che abbiamo riscontrato sono decisamente trascurabili, mentre le note positive sono innumerevoli.

Il Turbo Debugger è indubbiamente un prodotto d'alta classe, incredibilmente completo nonostante sia alla sua prima uscita.

Chi avesse già il Turbo C 2.0 o il Turbo Pascal 5.0, grazie alla sicura validità dei debugger in questi integrati, può anche fare a meno del debugger separato, ma un debugger separato come questo è certo una forte tentazione. Ancor più per chi fosse rimasto alle versioni precedenti dei due compilatori: difficile trovare il modo di non fare l'upgrade alle versioni «professional».

Quanto al prezzo, va considerato che la confezione comprende anche quel Signor Assembler esaminato il mese scorso, e con ciò dovremmo aver detto tutto.

MC