di Sergio Polini

Ridirezione dell'I/O Parametri della riga comando

Non si apprezza un linguaggio come il Pascal in programmi di poche decine di righe. Le liste sono strutture di dati flessibili e potenti, sicuramente sprecate per applicazioni del tipo «agenda» o «rubrica telefonica». Accade quindi che, tornando al tema delle strutture dinamiche di dati, ci imbattiamo subito nel problema degli esempi. Se ci si limita a programmini che stanno in una pagina si vede solo la punta dell'iceberg, non si può mostrare appieno l'uso di certe tecniche, non ci si può confrontare con lo stile di programmazione che il Pascal al tempo stesso consiglia e rende possibile.
Vi propongo pertanto una soluzione forse un po' insolita: costruire insieme un programma non banale, possibilmente anche utile, partendo dalla definizione del problema per arrivare fino alla implementazione degli ultimi dettagli; così facendo si potranno anche esaminare temi e problemi di carattere generale, utili quale che sia l'applicazione che si voglia realizzare.
Naturalmente questo comporta che non basteranno una o due puntate per arrivare alla meta, ma per fortuna ci viene in aiuto MC-Link: potete trovarvi sin da ora il programma completo

Il nostro obiettivo, come avevamo già sottolineato, non è solo di proporre esempi di uso di certe strutture di dati, ma anche di introdurre le tecniche di programmazione usate nei «toolbox» della Borland. È questo il motivo per cui costruiremo insieme un editor. Non un «full screen» (non vogliamo creare un doppione dell'Editor Toolbox), ma un editor «line oriented» analogo all'Edlin del DOS (forse più potente), e soprattutto analogo all'«edit» proposto da Kernighan e Plauger nel loro classico *Software Tools in Pascal*, nonché all'«ed» di UNIX (a cui lo stesso «edit» largamente si ispira). Un «quasi-ed», insomma, al quale daremo il nome di QUED. In realtà le differenze tra un editor «tutto schermo» ed uno «orientato alla riga» sono sensibili, tali da comportare anche nette differenze nella scelta delle strutture di dati più adatte; in particolare, Kernighan e Plauger propongono di usare un array di righe, mentre gli editor del Toolbox usano liste doppie. Visti i nostri fini, nella considerazione degli elementi pro e contro le possibili diverse scelte eserciterà un ruolo non trascurabile l'esigenza di «somiglianza» con il Toolbox; in questo modo avremo anche occasione di acquisire maggiore familiarità con le liste, dell'uso delle quali spero di potervi poi mostrare, subito dopo l'editor, altri esempi concreti.

Breve descrizione di QUED

Appena parte QUED si pone in attesa di un comando; non è quindi possibile cominciare subito a immettere il testo, ma bisogna prima avvertire il programma che le righe successive non conterranno comandi ma, appunto, il nostro testo. Questo si fa con una «a» (append) e si ritorna al «modo comando» con una linea rappresentata da «.» (un punto). È anche possibile lavorare su un testo creato in precedenza, indicandone il nome quando si fa partire il programma: con «QUED PIPPO», ad esempio il file PIPPO viene letto e caricato in memoria.

Per lavorare su un testo in memoria bisogna indicare la riga o le righe su cui si vuole eseguire un comando. Con «3d» (delete) si cancella la terza riga, con «2,30p» (print) si visualizzano le righe dalla seconda alla trentesima, con «4,7 m 10» (move) si spostano le righe 4, 5, 6 e 7 dopo la decima, e così via. Per facilitare l'indicazione di una riga si dispone di due caratteri convenzionali: un punto indica sempre la riga corrente (in genere l'ultima su cui è stato eseguito un comando), il simbolo del dollaro indica l'ultima; «.,$c» (change), ad esempio, cancella le righe comprese tra quella corrente e l'ultima e ci pone in modo «append»: quanto digiteremo do-

```
IF %1*==* GOTO NOBBS
ECHO g/%1-/p >BBS.CMD
:ANCORA
SHIFT
IF %1*==* GOTO FINE
ECHO g/%1-/p >>BBS.CMD
GOTO ANCORA
:FINE
QUED ELENCO.BBS <BBS.CMD >ELENCO.TMP
DEL BBS.CMD
:NOBBS
```

Figura 1
Il file BBS.BAT crea un file BBS.CMD contenente comandi di QUED. Chiama poi QUED in modo che questo legga il file ELENCO.BBS., esegua i comandi contenuti in BBS.CMD e invii a ELENCO.TMP l'output prodotto da questi comandi.

po il comando, fino a che non daremo il segnale di fine con una riga fatta di un solo punto, andrà a sostituire le righe cancellate. Una riga può essere identificata anche mediante una stringa in essa contenuta: con «/end/» ci posizioneremo sulla prima riga successiva a quella corrente contenente la stringa «end», con «?begin?» sulla prima riga precedente contenente «begin».

Il comando «s» (substitute) consente poi di sostituire una stringa con un'altra: «/mela/s//pera/p» cerca la prima riga contenente la stringa «mela», sostituisce la prima «mela» contenuta in quella riga con «pera» e visualizza poi la riga modificata. Per sostituire tutte le stringhe «mela» in una riga, si può usare «g» (global): «/mela/s//pera/gp». Il comando «g» può anche essere usato per ripetere un comando su tutto un insieme di righe; con «5,20g/mela/s//pera/g», ad esempio, si sostituiscono con «pera» tutte le «mela» contenute nelle righe tra la quinta e le ventesima.

Ci sono altri comandi e altre possibilità, che vedremo con completezza quando si tratterà di scrivere le relative routine. Ora possiamo magari solo accennare ai comandi «r» (read), che inserisce nel nostro testo il contenuto di un file, «w» (write), che scrive il testo su disco, e «q» (quit), che pone fine alle operazioni.

Normalmente non c'è nessuna differenza visibile tra l'immissione di un comando e di righe di testo; può quindi capitare, le prime volte, di non accorgersi di digitare comandi quando il programma si aspetta testo e viceversa. È comunque possibile farsi mostrare un «prompt» («>» per default, ma può essere cambiato) quando si è in modo comando. Ancora: se si commette un errore (ad esempio con «20,5p», che è sbagliato in quanto QUED non «gira attorno» alla fine del testo per visualizzare prima le righe dalla ventesima all'ultima e poi quelle tra la prima e la quinta), veniamo avvertiti con un semplice ed ermetico punto interrogativo; anche in questo caso è però possibile ottenere maggiore chiarezza e farsi mostrare un più esplicito messaggio d'errore.

A questo punto viene da chiedersi perché dover intervenire per correggere un ermetismo a prima vista inutile. Per dare una risposta dobbiamo però... ricominciare da zero, cioè ripercorrere dall'inizio la strada che ha condotto a far sì che QUED avesse le caratteristiche che ha.

```
program IORedir1;
{$G512,P512,D-}
{ Direttive G, P e D, procedura Flush(output) }
{ e "writeln(con,...)" per lo standard error  }
type
  s80 = string[80];
var
  s: string[80];
begin
  while not eof do begin
    readln(s);
    writeln(s); flush(Output);
  end;
  writeln(con,'Fine')  { non necessario Flush su CON: }
end.

program IORedir2;
{$D-}
{ File standard di I/O del DOS, direttiva D e procedura Flush }
type
  s80 = string[80];
var
  s: string[80];
  Inp, Out, Err: text;
begin
  assign(Inp,'INP:'); reset(Inp);
  assign(Out,'OUT:'); rewrite(Out);
  assign(Err,'ERR:'); rewrite(Err);
  while not eof(Inp) do begin
    readln(Inp,s);
    writeln(Out,s); flush(Out);
  end;
  writeln(Err,'Fine'); flush(Err);
end.

program IORedir3;
{ Come IORedir2, ma senza la direttiva D }
type
  s80 = string[80];
var
  s: string[80];
  Inp, Out, Err: text;
begin
  assign(Inp,'INP:'); reset(Inp);
  assign(Out,'OUT:'); rewrite(Out);
  assign(Err,'ERR:'); rewrite(Err);
  while not eof(Inp) do begin
    readln(Inp,s);
    writeln(Out,s); flush(Out)
  end;
  writeln(Err,'Fine'); flush(Err)
end.
```

Figura 2 - Variazioni sul tema della I/O redirection con il Turbo Pascal 3.0.

Ridirezione dell'I/O

Il lavoro di analisi preparatorio della stesura di un programma dovrebbe cominciare dalla individuazione delle caratteristiche delle tre fasi fondamentali: input, elaborazione, output. A prima vista si tratta di un lavoro piuttosto noioso e, nel caso specifico, in buona parte inutile. In fondo è scontato che per un editor l'input è dato dai comandi digitati e dal testo immesso (da tastiera o da un file), l'elaborazione dalla scrittura e modifica del testo, l'output dalla visualizza-

zione su video del testo e dalla sua eventuale memorizzazione in un file.

Ma non bisogna essere frettolosi, soprattutto si deve resistere alla tentazione di scrivere una sequela di «if... then... else.. else if... however... otherwise... but... who knows» ecc. Notiamo innanzitutto che «elaborazione» (come vedremo meglio il mese prossimo) non vuol dire solo «aggiunta di nuove righe a quelle già scritte», ma anche cancellazione o spostamento di brani, ricerca e sostituzione di stringhe, incorporazione nel testo in memoria di un altro testo da un file, ecc. Ma anche input e output sono meno scontati di quanto sembra.

L'MS-DOS, a partire dalla versione 2.0, ha fatto propri alcuni concetti di UNIX, in particolare la considerazione delle periferiche come file speciali (ma pur sempre tali) e le conseguenti possibilità di «I/O redirection» («DIR > DIRFILE» scrive la directory su un file invece che su video) e di «piping» («DIR | SORT» mostra una directory ordinata). Le funzioni del DOS che si incaricano dell'I/O accettano dati da uno «standard input» e li inviano ad uno «standard output»; se non si reindirizzano i canali standard, i dati arrivano dalla tastiera e vanno al video, altrimenti si può usare qualsiasi file su disco o qualsiasi periferica. Si può stampare un file con «TYPE PIPPO > PRN», si può creare un file «copiandolo» dalla tastiera con «COPY CON COMANDI», si può eseguire il DEBUG con comandi precedentemente memorizzati in un file digitando «DEBUG < COMANDI».

Un editor a tutto schermo non può giovarsi granché di queste possibilità, in quanto il suo funzionamento è fondamentalmente interattivo, guidato da quello che l'utente vede e dagli spostamenti del cursore sul video. Con un editor orientato alla riga le cose stanno invece diversamente: ci si posiziona su una certa riga, e si eseguono comandi sulla riga selezionata, semplicemente digitando opportune sequenze di caratteri sulla tastiera (e quindi questi caratteri possono anche venire da un file); l'output su video non comporta elaborazioni particolari (quali ad esempio la scrittura diretta sulla memoria video), ma deriva da semplici istruzioni di «write», come tali reindirizzabili a un file o alla stampante.

Supponiamo ad esempio che abbiate un file ASCII contenente i nomi di tutti i BBS italiani, i loro numeri di telefono (ad esempio: 06-4510211), i parametri di trasmissione (ad esempio: 300/1200 8-N-1). Di volta in volta avete bisogno di creare un file contenente i dati di tutti i BBS di una o più particolari città. Vi potete creare un piccolo file batch come quello in figura 1: digitando «BBS 02 06» ottenete un file ELENCO.TMP con i dati dei BBS di Milano e Roma, in quanto BBS.BAT usa QUED reindirizzandone sia l'input (da un file BBS.CMD che provvede prima a creare poi a cancellare) che l'output (inviato a ELENCO.TMP).

In altri termini, la ridirezione dell'I/O consente di costruire programmi «filtro», che inviano allo standard output il risultato di una elaborazione condotta sullo (o sulla base dello) standard input.

Notate che BBS.BAT non funzionerebbe a dovere se usassimo l'EDLIN invece di QUED. L'EDLIN infatti conosce *solo* standard input e standard output, non anche il terzo canale di I/O predefinito, lo standard error (sotto MS-DOS quest'ultimo non è reindirizzabile, in quanto coincide sempre con il video: viene infatti usato per mostrare messaggi d'errore o comunque altri dati che non vanno inviati all'eventuale file cui è reindirizzato lo standard ouput). Il risultato è che ELENCO.TMP conterrebbe, oltre ai BBS che cerchiamo, anche i comandi, il prompt, ecc., in genere tutto quello che vedremmo su video se usassimo EDLIN senza ridirezione.

```
program IORedir4;
uses Crt;
type
  s80 = string[80];
var
  s: string[80];
  StdErr: text;
begin
  assign(Input,''); reset(Input);
  assign(Output,''); rewrite(Output);
  assigncrt(StdErr); rewrite(StdErr);
  while not eof do begin
    readln(s);
    writeln(s);
  end;
  writeln(StdErr,'Fine');
end.
```

Figura 3
Esempio di I/O redirection in Turbo Pascal 4.0.

INP:, OUT:, ERR:, $G, $P, $D

Il Turbo Pascal usa spesso il BIOS per l'I/O, e questo rende impossibile la ridirezione. Con la versione 3.0, tuttavia, si dispone di vari strumenti per poter sfruttare anche questa possibilità offerta dal DOS. Il manuale comunque non è molto chiaro al riguardo: sembra infatti che l'unica soluzione consista nell'impiego congiunto delle direttive «G» (per l'input), «P» (per l'output) e «D»; l'unico programma proposto come esempio a pagina 202 del manuale comincia appunto con {$G512, P512, D-}, e il testo spiega che: a) le prime due direttive richiedono un argomento intero che definisca la dimensione del buffer di I/O, e solo se questo argomento è maggiore di zero verranno usati lo standard input e lo standard output del DOS; b) la direttiva D, attiva per default, controlla che un file aperto con Reset, Rewrite o Append sia un file «vero», e se riscontra che si tratta invece di un file «speciale» (cioè una periferica) disabilita la bufferizzazione; va quindi disattivata ogni volta che si usano le altre due direttive con dimensioni di buffer maggiori di zero.

Per prima cosa nulla si dice sullo standard error (ma basta usare istruzioni del tipo «write (con, ...)» per mandare sempre e solo al video quello con cui non si voglia eventualmente «sporcare» lo standard output). Ma soprattutto non si chiarisce che quello non è il solo modo per attivare la I/O redirection: si può infatti tranquillamente fare a meno delle tre direttive appena viste, usando i file speciali INP:, OUT: e ERR: (li trovate a pagina 200 del manuale) e ricordandosi di chiamare la procedura Flush dopo i Write al fine di provocare lo svuotamento del buffer di output. Potete trovare nella figura 2 diversi esempi di programmi «reindirizzabili»; provate a compilarli su disco e a eseguirli digitando, ad esempio, «IOREDIR1 <IOREDIR1.PAS >TEMP».

Il Turbo Pascal 4.0 ha finalmente messo le cose a posto. Il compilatore produce per default codice che, usando il DOS per l'I/O, consente sempre di reindirizzare l'input e l'output. Se si usa la Unit CRT si ottengono numerosi vantaggi (output su video più veloce, finestre, colore, controllo dei codici «estesi» della tastiera, ecc.), ma i file predefiniti Input e Output non

Figura 4
Una prima versione della routine di inizializzazione di QUED.

```
procedure Q_Inizializza(var Stato: integer);
var
  i, j, lps: integer;
  ps: AnyStr;
begin
  assign(StdInp,'INP:'); reset(StdInp);
  assign(StdOut,'OUT:'); rewrite(StdOut);
  assign(StdErr,'ERR:'); rewrite(StdErr);
  i := 1; Stato := OK;
  ErroriInChiaro := FALSE; MostraPrompt := FALSE; MostraTotRighe := TRUE;
  PromptStr := '>'; NomeFile := '';
  while (i <= ParamCount) and (Stato = OK) do begin
    ps := ParamStr(i); lps := length(ps);
    if ps[1] = '/' then begin
      for j := 2 to lps do
        if ps[j] = 'h' then ErroriInChiaro := TRUE
        else if ps[j] = 's' then MostraTotRighe := FALSE
        else if (ps[j] = 'p') and (j = lps) and (ParamCount > i) then begin
          MostraPrompt := TRUE; i := i + 1; PromptStr := ParamStr(i)
        end
        else Stato := ERRPARAM
    end
    else if NomeFile = '' then NomeFile := ps
    else Stato := ERRPARAM;
    i := i + 1
  end;
  if (Stato = OK) and (NomeFile <> '') then
    LeggiFile(NomeFile, Stato)
end;
```

vengono più associati ai corrispondenti «standard» del DOS. Si può comunque facilmente rimediare con le istruzioni

```
Assign (Input, ''); Reset(Input);
Assign (Output, ''); Rewrite(Output);
```

che annullano in parte l'effetto dell'uso della Unit CRT, facendo sì che il programma compilato usi lo standard input e lo standard output per le operazioni di I/O.

È anche possibile usare uno standard error usando la procedura AssignCrt contenuta nella Unit:

```
AssignCrt(StdErr); Rewrite(StdErr);
```

assegna CRT (cioè il video) a StdErr, che va dichiarato come file di tipo Text.

ParamCount e ParamStr

Riepiloghiamo: se si ha cura di distinguere in qualche modo tra standard output e standard error, uno dei vantaggi di un editor orientato alla riga consiste nella possibilità di usarlo in un file batch, di avvalersene come «filtro» per ottenere in modo veloce e automatico un output che sia il risultato di elaborazioni, anche complesse, su un dato input.

Questo comporta anche che il nostro editor deve poter funzionare in almeno due modi: uno «chiacchierone» ed uno «silenzioso». Quando lo usiamo direttamente può esserci utile avere pronto riscontro del risultato dei comandi che gli diamo, quando lo facciamo eseguire da un file batch (o quando ce ne sentiamo ormai padroni) possiamo preferire che non ci mandi continuamente messaggi di conferma o di errore.

Definiamo quindi alcune variabili boolean:

a) ErroriInChiaro: se vera mostra un messaggio esplicativo ogni volta che si verifica un errore, se falsa (default) mostra solo un punto interrogativo.

b) MostraPrompt: se vera mostra un «prompt» quando si è in «modo comandi», se falsa (default) sta a noi riconoscere in che modo siamo.

c) MostraTotRighe: se vera (default), ogni volta che legge da o scrive su un file mostra il numero di righe lette o scritte, se falsa opera «in silenzio» non dando alcuna conferma dell'avvenuta lettura/scrittura.

Alcuni comandi (rispettivamente H, P e S) consentono di cambiare il valore di queste variabili mentre si sta usando il programma, ma, se vogliamo controllare l'esecuzione anche quando il programma è eseguito da una file batch, dobbiamo poter effettuare le nostre scelte nel momento stesso in cui il programma viene attivato. Dobbiamo anche poter indicare nello stesso modo su quale file il programma deve lavorare.

Quando digitiamo il nome di un programma per eseguirlo, possiamo far seguire al nome alcuni parametri preceduti o meno da una barra («/»). Il Turbo Pascal ci mette a disposizione due funzioni predefinite che sono di grande aiuto nella interpretazione di questi parametri: ParamCount ritorna il loro numero, ParamStr(i) ritorna l'i-esimo parametro (ParamStr(1) il primo, ParamStr(2) il secondo, ecc.).

Possiamo così costruire finalmente il primo pezzo di QUED. Nella figura 4 c'è una versione provvisoria di Q_Inizializza, la routine eseguita appena parte il programma: vengono aperti i file «speciali», viene assegnato il valore di default alle variabili boolean che abbiamo appena visto, vengono finalmente interpretati i vari parametri: «/h» abilita i messaggi d'errore in chiaro, «/p qualcosa» fa si che, quando siamo in modo comandi, ci venga mostrato questo <qualcosa> come prompt «/s» rende «silenziose» le operazioni di lettura/scrittura di file, una stringa non preceduta da una barra (e nemmeno da «/p») viene interpretata come il nome del file su cui si vuole lavorare. Notate che Q_Inizializza è costruita in modo tale che dopo una barra vi possono essere anche più parametri, ad esempio:

```
qued Pippo /hsp Comando:
```

Per ora è tutto. La prossima volta vedremo come scegliere le strutture di dati più adatte al nostro editor.

MC