Un po' di EDITORS e un po' di TVPRINT

Un lettore, Fabrizio Superchi di Brembate (BG), mi ha scritto per propormi le sue soluzioni di quattro bug che ha trovato nella unit EDITORS. Ho avuto modo di riscontrare che due di essi sono stati eliminati nella nuova versione della unit (quella che accompagna il Borland Pascal 7.0). Vedremo quindi in dettaglio gli altri due, accennando brevemente a quelli ormai risolti a beneficio di coloro che ancora non avessero aggiornato il loro compilatore

di Sergio Polini

I bug individuati da Fabrizio Superchi sono di due tipi. Il primo consiste nel malfunzionamento dei tasti 3, 4, 5 e 6 della tastiera e di tutti i tasti tranne il 5 del tastierino numerico. Si tratta di un problema di cui si era ampiamente discusso nell'area PASCAL di MC-Link, ma di cui non avevo avuto ancora modo di riferire su queste pagine. Fabrizio ne propone una sua soluzione, alla quale preferisco però quella, più semplice, già individuata da Tommaso Masi (che ricorderete autore di una serie di articoli su Smalltalk); è sufficiente, infatti, intervenire sulla prima istruzione OR nella funzione *ScanKeyMap*, sostituendo OR BL,BL con OR DL,DL (figura 1).

Fabrizio ha comunque scoperto altri tre bug piuttosto insidiosi, tutti relativi alla ricerca di una stringa nel testo. Il peggiore di tutti era contenuto nella funzione *IScan*, utilizzata per ricerche insensibili alla differenza tra lettere maiuscole e minuscole: cercando una stringa in un testo che terminasse con una parola il cui primo carattere fosse uguale al primo della stringa cercata (ad esempio, cercando «rose» in un testo che terminasse con «una bella rosa»), si rischiava il loop infinito. Dopo un MOV CX,DX, infatti, compariva un salto condizionato JNE, come se l'autore della routine avesse dimenticato che l'istruzione MOV non modifica i flag; aggiungendo un'istruzione OR tutto torna a posto (figura 2).

La funzione Scan

Eccoci ora ai due bug che rimangono anche nell'ultima versione della unit.

Il primo si verifica quando, effettuando una ricerca sensibile alla differenza tra lettere maiuscole e minuscole, si cerca una stringa che compaia nel testo preceduta da una ripetizione del suo carattere iniziale; in altri termini, quando si cerca «ciao» in «xxx cciao yyy», la stringa non viene trovata.

Il bug si trova nella funzione *Scan* della unit EDITORS.PAS. Questa si avvale delle istruzioni SCASB e CMPSB

```
@@1: LODSW                          @@1: LODSW
     MOV  BX,AX                          MOV  BX,AX
     LODSW                               LODSW
     OR   BL,BL        --------->        OR   DL,DL
```

Figura 1 - Correzione del bug contenuto nella funzione ScanKeyMap *della unit EDITORS.PAS.*

```
     MOV  CX,DX                          MOV  CX,DX
     JNE  @@4                            OR   CX,CX
@@8: XOR  AX,AX        --------->        JNE  @@4
                                    @@8:  XOR  AX,AX
```

Figura 2 - Correzione del bug contenuto nella funzione IScan *della unit EDITORS.PAS.*

del microprocessore: con REPNE SCASB viene cercato nel testo il primo carattere della stringa; se non viene trovato, sicuramente la stringa non è nel testo e si esce, altrimenti si usa REP CMPSB per verificare se i caratteri successivi nel testo sono uguali a quelli della stringa che seguono il primo. Per poter usare tali istruzioni, viene posto in ES:DI l'indirizzo della regione del testo in cui svolgere la ricerca e in DS:SI l'indirizzo della stringa da cercare; poiché quelle istruzioni incrementano sia DI che SI, se il test con REP CMPSB dà esito negativo occorre ripetere il primo test con REP SCASB dopo aver riportato indietro i registri DI e SI (si aggiunge loro un CX con un valore negativo). Purtroppo, dopo il decremento, DI viene inopinatamente incrementato di nuovo con una istruzione INC DI; ne segue che la «c» di «ciao» non viene cercata a partire da «ciao yyy», ma a partire da «iao yyy» e, quindi, non viene trovata. Le istruzioni prima della label @@3 vanno pertanto corrette come indicato nella figura 3.

Ciò non basta, in quanto la ricerca non avrebbe ancora successo se il testo terminasse proprio con «cciao». In DX, infatti, viene posta la lunghezza del testo in cui cercare, che all'inizio è pari al numero dei caratteri compresi tra quello da cui inizia la ricerca e l'ultimo, diminuito della lunghezza della stringa da cercare (non avrebbe senso cercare una

stringa di quattro caratteri in un testo di tre). All'inizio della label @@2, DX viene assegnato a CX per determinare il numero di ripetizioni dell'istruzione REPNE SCASB; dopo di questa, si esegue l'assegnazione inversa (MOV DX,CX), mediante la quale, essendo stato CX decrementato da REPNE SCASB, si pone in DX la lunghezza del testo in cui ancora non si è cercato. Purtroppo, la prima istruzione del blocco con label @@2 è DEC DX, che altera il conto. In concreto succede che, dopo aver trovato la prima «c» in «cciao», DX vale correttamente 1 (ES:DI punta alla seconda «c» e REPNE SCASB va eseguita una sola volta, su quest'ultimo carattere, in quanto una «c» che fosse in terzultima o penultima o ultima posizione non potrebbe essere l'inizio di un «ciao» lungo quattro caratteri); dopo DEC DX, il registro DX contiene zero e, quindi, SCASB non viene eseguita e la stringa non viene trovata.

Il codice va modificato come indicato nella figura 4.

Fabrizio propone anche un'interessante alternativa a *TBlackFrame*; mi fa notare, infatti, che potrebbe non essere necessario usare una piccola folla di *TStaticText*; sarebbe infatti sufficiente inserire nella dialog box una *TView* con opzione *ofFramed*. In concreto, sarebbe sufficiente un codice analogo a quello riprodotto nella figura 5.

Gli unici inconvenienti sono di carattere cromatico: si ottiene infatti un frame dello stesso colore del frame della dialog box, quindi normalmente bianco, o verde quando la dialog box viene spostata sullo schermo. Se però vi basta un *TWhiteFrame*... perché no?

La classe TPrSelectDialog

Torniamo ora alla unit TVPRINT. La volta scorsa abbiamo visto la sua interfaccia, con la dichiarazione della classe *TPrinter*, nonché costanti, tipi e variabili dichiarati nella sua sezione **implementation;** vedremo ora l'implementazione di alcune classi utilizzate da *TPrinter*.

Nulla da dire per la classe *TPrinterCollection*; trattandosi di una collezione di stringhe non derivate da *TObject*, è necessario unicamente ridefinire il metodo *FreeItem* (figura 6).

La classe *TPrSelectDialog* (figura 7) viene utilizzata dal metodo *TPrinter.DoSetup* per costruire una dialog box con cui proporre all'utente la scelta tra una delle stampanti installate e consentirgli di cambiarne l'impostazione. In un programma possono esservi più istanze di *TPrinter*, così come ad un computer possono essere interfacciate più stampanti; il constructor richiede quindi un parametro *Prn* di tipo *PPrinter*, il cui valore viene assegnato alla variabile d'istanza *PP*.

Durante l'inizializzazione viene scorsa la directory in cui si trovano i file di definizione delle stampanti installate (directory sulla quale, nel metodo *TPrinter.DoSetup*, ci si è spostati con la funzione *ChangeToTVPRNDIR*); i singoli file vengono letti per ricavarne il nome della stampante e la porta di output e per costruire quindi delle stringhe descrittive, che vengono aggiunte alla *TPrinterCollection* puntata dalla variabile d'istanza *PrList*. Nel fare ciò, si prende nota del numero d'ordine del file corrispondente alla stampante correntemente selezionata (quella del parametro *Prn*), per poi costruire una list box in cui questa risulti evidenziata.

La dialog box conterrà anche un pulsante «Imposta» mediante il quale l'utente potrà intervenire sulla stampante che risulterà selezionata nella list box. Nel metodo *HandleEvent* viene dedicata particolare attenzione ai comandi *cmOk* e *cmSetup*. In entrambi i casi, viene innanzitutto ricostruito il nome del file contenente la descrizione della stampante selezionata nella list box *(SelPrn)*; se l'utente ha premuto il pulsante «Ok», quel file viene letto e la definizione assegnata al campo *PD* della *TPrinter* puntata da *PP*. Se l'utente ha premuto il pulsante «Imposta», alla lettura del file segue l'apertura di una dialog box istanza della classe *TPrSetupDialog* (dichiarata nella unit PRSETUP); nel caso questa venga chiusa con il

```
    ADD  SI,CX                          ADD  SI,CX
    ADD  DI,CX                          ADD  DI,CX
    INC  DI              ---------->    OR   DX,DX
    OR   DX,DX                          JNE  @@2
    JNE  @@2                        @@3: XOR  AX,AX
@@3: XOR  AX,AX
```

Figura 3 - Correzione del primo dei due bug contenuti nella funzione Scan *della Unit EDITORS.PAS.*

```
    INC  DX
    INC  DX                             INC  DX
@@2: DEC  DX             ---------->    MOV  CX,DX
    MOV  CX,DX                          REPNE SCASB
    REPNE SCASB
```

Figura 4 - Correzione del secondo bug contenuto nella funzione Scan *della unit EDITORS.PAS.*

```
var
  R: TRect;
  V: PView;
  ...
begin
  ...
  R.Assign(x1,y1,x2,y2);
  V := New(PView, Init(R));
  V^.Options := V^.Options or ofFramed;
  Insert(V);
  ...
```

Figura 5 - Un'alternativa «economica» alla classe TBlackFrame.

```
procedure TPrinterCollection.FreeItem(Item: Pointer);
begin
  DisposeStr(PString(Item));
end;
```

Figura 6 - L'implementazione della classe TPrinterCollection.

MDI, MC-Link, Internet e tutti voi

Da qualche tempo, MC-Link offre ai suoi abbonati un'opportunità di notevole interesse. Grazie alla connessione con Internet, è possibile comunicare con chiunque, nel mondo, abbia un indirizzo su quella rete. È stato così, ad esempio, che ho potuto contattare Andrzej Resztak, un polacco autore di alcune interessanti unit per la stampa di grafici realizzati con la BGI (il suo indirizzo Internet è: Resztak@PLUMCS11.bitnet).

Mi è però anche successo che Roberto Soldi, uno studente milanese di ardente temperamento, abbia approfittato di Internet per accennare, sia pure solo per scartarla, all'ipotesi di un «duello» per «lavare l'onta» dell'articolo del gennaio 1992, in cui avevo illustrato alcune unit con le quali aggiungevo un ribbon ed una riga di stato alle classi che, in ObjectWindows, sono dedicate alla realizzazione di applicazioni MDI.

Tanta veemenza è stata causata dai problemi da lui incontrati con quelle unit.

Per quanto mi risulta, quelle unit, compilate con il Turbo Pascal per Windows 1.0 sotto Windows 3.0, funzionavano e funzionano perfettamente anche sotto Windows 3.1 (come mi hanno confermato anche alcuni di voi che, a suo tempo, le hanno usate in loro programmi).

Devo riconoscere, tuttavia, che se compilate col nuovo Borland Pascal 7.0 sotto Windows 3.1, quelle stesse unit sono... un vero disastro! Ho potuto verificare anche troppo agevolmente gli inconvenienti lamentati da Roberto Soldi (basta provare a far sparire il ribbon).

Evidentemente è cambiato qualcosa, e si richiedono interventi sul codice.

Roberto mi ha accennato alle sue soluzioni, aggiungendo però che non poteva mandarmi i listati in quanto fanno parte della sua tesi di laurea. Nulla da eccepire.

Quanto a me, tra impegni di vario genere e il lavoro già programmato per la rubrica, non ho ora il tempo di tornare sull'argomento. Potrebbe però venire voglia a qualcuno di voi di dare un'occhiata a quelle unit, tentare una propria soluzione, e inviarmela per posta o tramite MC-Link o Internet perché io possa metterla a disposizione di tutti.

Chi raccoglie il guanto?

```
constructor TPrSelectDialog.Init(var Bounds: TRect; Prn: PPrinter);
var
  PD: TPrinterDef;
  DirInfo: SearchRec;
  S: PScrollBar;
  Def, N: Integer;
  R: TRect;
begin
  TDialog.Init(Bounds, '');
  Options := Options or ofCentered;
  PP := Prn;
  New(PrList, Init(5,5));
  Def := 0;
  N := -1;
  FindFirst('*.PRN', Archive, DirInfo);
  while DosError = 0 do begin
    Inc(N);
    if ReadPrinterInfo(DirInfo.Name, PD) then begin
      PrList^.Insert(NewStr(PD.Name+' ('+PD.FName+') su '+PD.Setup.OutPort));
      if PD.FName = PP^.PD.FName then Def := N;
      Dispose(PD.Setup.FontList, Done);
    end;
    FindNext(DirInfo);
  end;
  R.Assign(55,3,56,10);
  S := New(PScrollBar, Init(R));
  Insert(S);
  R.Assign(3,3,55,10);
  LB := New(PListBox, Init(R, 1, S));
  LB^.NewList(PrList);
  LB^.FocusItem(Def);
  Insert(LB);
  R.Assign(3,2,13,3);
  Insert(New(PLabel, Init(R, 'Stam-p-anti', LB)));
  R.Assign(57,3,71,5);
  Insert(New(PButton, Init(R, 'O-k-', cmOk, bfDefault)));
  R.Assign(57,5,71,7);
  Insert(New(PButton, Init(R, 'Annulla', cmCancel, bfNormal)));
  R.Assign(57,9,71,11);
  Insert(New(PButton, Init(R, '-I-mposta...', cmSetup, bfNormal)));
  SelectNext(False);
end;

destructor TPrSelectDialog.Done;
begin
  LB^.NewList(nil);
  TDialog.Done;
end;
```

```
procedure TPrSelectDialog.HandleEvent(var Event: TEvent);
var
  SelPrn: String;
  i,j: Integer;
  D: PPrSetupDialog;
  R: TRect;
begin
  if Event.What = evCommand then
    case Event.Command of
      cmOk,
      cmSetup: begin
                 SelPrn := PString(LB^.List^.At(LB^.Focused))^;
                 i := Length(SelPrn);
                 while SelPrn[i] <> '(' do
                   Dec(i);
                 Inc(i);
                 j := i;
                 while SelPrn[j] <> ')' do
                   Inc(j);
                 SelPrn := Copy(SelPrn, i, j-i) + '.PRN';
                 case Event.Command of
                   cmOk: if ReadPrinterInfo(SelPrn, PR) then begin
                           if PP^.PD.Setup.FontList <> nil then
                             Dispose(PP^.PD.Setup.FontList, Done);
                           PP^.PD := PR;
                         end
                         else
                           MessageBox('Errore lettura file '+SelPrn,
                             nil, mfError or mfOkButton);
                   cmSetup: if ReadPrinterInfo(SelPrn, PR) then begin
                              R.Assign(1,2,79,22);
                              D := New(PPrSetupDialog, Init(R));
                              D^.SetData(PR.Setup);
                              if DeskTop^.ExecView(D) = cmOk then begin
                                D^.GetData(PR.Setup);
                                WritePrinterInfo(SelPrn, PR);
                                LB^.List^.AtFree(LB^.Focused);
                                LB^.List^.AtInsert(LB^.Focused,
                                  NewStr(PR.Name+' ('+PR.FName+') su '+PR.Setup.OutPort))
                              end
                              else
                                Dispose(PR.Setup.FontList, Done);
                              Dispose(D, Done);
                              Redraw;
                            end
                            else
                              MessageBox('Errore lettura file '+SelPrn,
                                nil, mfError or mfOkButton);
                 end;
               end;
    end;
  TDialog.HandleEvent(Event);
end;
```

Figura 7 - L'implementazione della classe TPrSelectDialog.

```
constructor TFontSelectDialog.Init(var Bounds: TRect; AFont: Integer;
  Prn: PPrinter);
var
  V: PView;
  R: TRect;
begin
  TDialog.Init(Bounds, 'Tipi carattere');
  Options := Options or ofCentered;
  PP := Prn;
  R.Assign(75,2,76,6);
  V := New(PScrollBar, Init(R));
  Insert(V);
  R.Assign(2,2,75,6);
  FB := New(PFontBox, Init(R, PScrollBar(V)));
  FB^.NewList(PP^.PD.Setup.FontList);
  if FB^.List <> nil then begin
    if AFont >= FB^.Range then
      AFont := FB^.Range - 1;
    FB^.FocusItem(AFont);
  end;
  Insert(FB);
  R.Assign(24,7,35,9);
  Insert(New(PButton, Init(R, 'O-k-', cmOk, bfDefault)));
  R.Assign(45,7,56,9);
  Insert(New(PButton, Init(R, 'Annulla', cmCancel, bfNormal)));
  SelectNext(False);
end;

procedure TFontSelectDialog.HandleEvent(var Event: TEvent);
begin
  if (Event.What = evBroadcast) and (Event.Command = cmListItemSelected)
  then begin
    Event.What := evCommand;
    Event.Command := cmOk;
  end;
  TDialog.HandleEvent(Event);
end;
```

```
constructor TPrintFileDialog.Init(var Bounds: TRect);
var
  R: TRect;
begin
  TDialog.Init(Bounds, 'Stampa su file');
  R.Assign(2,3,40,4);
  FName := New(PInputLine, Init(R, 64));
  Insert(FName);
  R.Assign(2,2,22,3);
  Insert(New(PLabel, Init(R, '-N-ome file di output', FName)));
  R.Assign(42,2,53,4);
  Insert(New(PButton, Init(R, 'O-k-', cmOk, bfDefault)));
  R.Assign(42,4,53,6);
  Insert(New(PButton, Init(R, 'Annulla', cmCancel, bfNormal)));
  SelectNext(False);
end;


function TPrintFileDialog.Valid(Command: Word): Boolean;
var
  DirInfo: SearchRec;
  OkToCreate: Boolean;
begin
  if TDialog.Valid(Command) then begin
    if Command = cmOk then begin
      if FName^.Data^[0] = #0 then
        Valid := False
      else begin
        OkToCreate := True;
        FindFirst(FName^.Data^, Archive, DirInfo);

        if DosError = 0 then
          if MessageBox(#3'Il file già esiste.'#13#3'Confermi?', nil,
              mfYesButton or mfNoButton or mfConfirmation) <> cmYes then
            OkToCreate := False;
        Valid := OkToCreate;
      end;
    end;
  end
  else
    Valid := False;
end;
```

Figura 8 - L'implementazione delle classi TFontSelectDialog *e* TPrintFileDialog.

pulsante «Ok», vengono aggiornati sia il file su disco che la stringa descrittiva nella list box.

Il meccanismo è concepito in modo da consentire all'utente di cambiare l'impostazione di qualsiasi stampante installata sul suo sistema, anche di una che non intenda utilizzare subito dopo aver chiuso la dialog box.

La classe TFontSelectDialog

Una volta selezionata una stampante, diventa immediatamente accessibile l'elenco dei tipi di carattere disponibili, in quanto «puntato» dal campo *Setup.FontList* della variabile *PD* dell'istanza di *TPrinter* passata al constructor di *TPrSelectDialog*.

Per scegliere un tipo di carattere, il metodo *TPrinter.SelectFont* si avvale di una dialog box istanza della classe *TFontSelectDialog* (figura 8). Anche il constructor di questa richiede, tra gli altri, un parametro *Prn* attraverso il quale passare un puntatore alla *TPrinter* in uso; oltre a questo, compare anche un parametro *AFont*, mediante il quale si informa la dialog box del tipo di carattere correntemente selezionato. Si procede, infatti, alla costruzione di una list box con l'elenco dei tipi di carattere disponibili, tra i quali quello corrente risulti evidenziato.

Come vedremo, *TPrinter.SelectFont* usa la variabile d'istanza *Font* come parametro attuale; potrebbe succedere che, una volta selezionata una stampante diversa da quella di default o appena usata, *Font* abbia un valore superiore al numero dei tipi di carattere divenuti disponibili, con ovvi inconvenienti. Avrei potuto scegliere di reinizializzare a zero la variabile *Font* dopo ogni selezione di stampante, ma ho preferito effettuare un controllo ed eventualmente correggere il suo valore se non più adeguato. Una semplice questione di gusti.

Quanto al metodo *HandleEvent*, va solo sottolineata l'intercettazione del comando *cmListItemSelected*. Questo viene generato ogni volta che, in un oggetto appartenente ad una classe derivata da *TListViewer*, l'utente seleziona un elemento della lista con un doppio click del mouse o premendo la barra spaziatrice; intercettandolo e trasformandolo in un comando *cmOk* si evita che l'utente, una volta scelto un tipo di carattere, debba agire sul pulsante «Ok».

La classe TPrintFileDialog

La figura 8 comprende anche l'implementazione della classe che viene usata quando, dato avvio alla stampa, questa risulta indirizzata su file invece che su una porta seriale o parallela.

Viene aperta una dialog box con la quale si chiede il nome del file di output. Il metodo *Valid* si incarica di verificare se un file con quel nome già esista, nel qual caso viene chiesta conferma.

Per ora ci fermiamo qui. Il mese prossimo vedremo, finalmente, l'implementazione della classe *TPrinter*.

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