Appendix A Een module TextWindows voor MS-DOS

De eerste drie voorbeeldprogramma's uit het boek zijn met elke Modula 2 compiler op een willekeurige computer te compileren en uit te voeren. Bij de programma's Parser en EBNF wordt invoer van het toetsenbord verwacht. Bij General Parser wordt de invoer uit een bestand op disk gelezen. De extensie van dat bestand wordt gedefinieerd als 'EBNF'.
Het invoerbestand moet onder MS-DOS de extensie .EBN hebben; de laatste F zal automatisch door MS-DOS worden 'weggegooid'.
Voor het PL/O-systeem is een module TextWindows nodig. Deze module moet ervoor zorgen dat de verschillende uitvoerstromen van het systeem in verschillende vensters op het beeldscherm worden afgedrukt. In deze bijlage vindt u zo'n module voor MS-DOS. Deze is gemaakt voor de Modula-2/86 compiler van Logitech; gebruik van de module vereist dat de Window Machine van die compiler in het geheugen is geladen.

In de module worden alle door het PL/0-systeem uit de module TextWindows geimporteerde procedures en variabelen voor export gedefinieerd. Een aantal daarvan worden daartoe eenvoudigweg weer uit de module WM (Window Machine) geimporteerd.
Het programma spreekt voor zich. In de procedure OpenTextWindow wordt er via een wat eigenaardig algoritme voor gezorgd dat de vensters op een beeldscherm van 80 x 25 passen.
In de module CloseTextWindow wordt bij het sluiten van elk venster de gelegenheid geboden de inhoud van het venster nogmaals te bekijken: met de toetsen UP en DOWN wordt naar de vorige, respectievelijk volgende regel gegaan, met de toetsen Home en End naar het begin, respectievelijk einde van het venster.
Deze toetsen om de cursor te verplaatsen werken alleen als de ANSI-driver is geladen. (In CONFIG.SYS moet de opdracht device=ansi.sys staan; ANSI.SYS moet uiteraard op disk te vinden zijn.) Met Escape wordt het venster gesloten en het volgende geopend. Als alle vensters gesloten zijn stopt het programma.

De programmamodules in het boek zijn uiteraard gebaseerd op de laatste versie (1983) van de definitie van Modula-2. In deze versie is de clausule EXPORT QUALIFIED niet langer nodig. Bij de door mij voor deze bijlage gebruikte versie van de Modula2/86 compiler van Logitech zijn deze clausules nog wel vereist. Dit betekent dat in de definitiemodules van het PL/0-systeem aan het begin -na een eventueel IMPORT-statement- een opsomming moet worden gegeven van de uit de betreffende module exporteerbare typen, variabelen, procedures, enzovoort. Gemakshalve zetten we deze clausules hieronder nog even per definitiemodule op een rijtje.

In de definitiemodule van PL0Scanner (programma 4, eerste deel) moet onder het FROM-statement worden ingevoegd:

```
EXPORT QUALIFIED  Symbol, sym, id, num, Diff, KeepId, GetSym, Mark,
                  InitScanner, source, CloseScanner;
```

In de definitiemodule van PL0Parser (programma 5, eerste deel) moet helemaal aan het begin worden ingevoegd:

```
EXPORT QUALIFIED Parse, noerr, EndParser;
```

In de definitiemodule van PL0Interpreter (programma 6, eerste deel) moet helemaal aan het begin worden ingevoegd;

```
EXP0RT QUALIFIED  Instruction, maxfct, maxadr, code, Interpret,
                  EndInterpreter;
```

In de definitiemodule van PL0Generator (programma 7, eerste deel) moet helemaal aan het begin worden ingevoegd;

```
EXPORT QUALIFIED Label, Gen, fixup, InitGenerator, EndGenerator;
```

Ook in de in deze bijlage opgenomen module TextWindows staat zo'n clausule. In de volgende versie van Modula2/86 zullen deze clausules niet meer nodig zijn.

Een diskette met de programmatekst van alle voorbeelden en een werkende versie van het PL/0-systeem is verkrijgbaar bij Academic Service; zie voor de bij bestelling van deze diskette te volgen procedure de desbetreffende mededeling voor in dit boek (pagina ii).

```
DEFINITION MODULE TextWindows;

FROM  WM    IMPORT  Screen;

EXPORT QUALIFIED  Window, Done, Write, WriteString, WriteLn, WriteCard,
                  WriteInt, ReadInt, Invert, 0penTextWindow,
                  CloseTextWindow;

TYPE  Window      = Screen;

VAR   Done        : BOOLEAN;

PROCEDURE Write (win : Window; ch: CHAR);
PROCEDURE WriteString (win : Window; s : ARRAY OF CHAR);
PROCEDURE WriteLn (win : Window);
PROCEDURE WriteCard (win : Window; card, len: CARDINAL);
PROCEDURE WriteInt (win : Window; int : INTEGER; len : CARDINAL);
PR0CEDURE ReadInt (win : Window; VAR int : INTEGER);
PRQCEDURE Invert (win : Window; inv: BOOLEAN);
PROCEDURE OpenTextWindow (VAR win : Window; x, y, xpixels, ypixels:
CARDINAL; text: ARRAY DF CHAR);
PROCEDURE CloseTextWindow (win : Window);
END TextWindows.
```

```
IMPLEMENTATION MDDULE TextWindows;

IMPORT  WM, In0ut;

FROM  SYSTEM      IMP0RT  ADR, SIZE;
FROM  Strings     IMPORT  Concat, Insert;
FROM  Conversions IMP0RT  ConvertCardinal, ConvertInteger;

CONST VBufSize    = 65500;
```

```
TYPE  STRING      = ARRAY [0..80] OF CHAR;
      String      = ARRAY [O..8 ] OF CHAR;

VAR   rc          : CARDINAL;
      VBuf        : ARRAY [O..VBufSize-1] OF CHAR;
      winH        : Window;

PR0CEDURE Invert (win : Window; inv : BOOLEAN);

BEGIN
  IF  inv = TRUE  THEN
    rc := WM.SetDefaultAttribute (win, WM.reverseLow);
  ELSE
    rc := WM.SetDefaultAttribute (win, WM.normalLow)
  END
END Invert;


PRDCEDURE ReadInt (win: Window; VAR int; INTEGER);

BEGIN
  rc := WM.OpenWindow (win);
  Inout.ReadInt (int);
  Done := In0ut.Done;
  WriteInt (win, int, 17;
  Invert (win, FALSE);
  WriteLn (win)
END ReadInt;


PROCEDURE WriteInt (win: Window; int: INTEGER; len: CARDINAL);

VAR     s  : string;

BEGIN
  ConvertInteger (int, len, 8);
  rc := WM.Write (win, 8)
END WriteInt;


PROCEDURE WriteCard (win; Window; card, len; CARDINAL);

VAR   s, s1             : string;
      row, col, count   : CARDINAL;

BEGIN
  ConvertCardinal (card, len, s);
  rc := WM.ReadScreenText (win, O, O,  1, s1);
  IF  s1 [O] <> 'R'  THEN  Insert (' ', s, 0)  END;
  rc := WM.Write (win, s)
END WriteCard;


PROCEDURE WriteString (win : Window; s : ARRAY OF CHAR);
```

```
BEGIN
  rc := WM.OpenWindow (win);
  rc := WM.Write (win, s)
END WriteString;



PROCEDURE WriteLn (win : Window);

VAR   s     : string;

BEGIN
  s[0] := 15C;  s[1] := 12C; s[2] := 0C;
  WriteString (win, s);
END WriteLn;


PROCEDURE Write (win : Window; ch : CHAR);

VAR   s : string;

BEGIN
  CASE  ch  0F
   14C : rc := WM.WriteAt (win, 1, 0,  '~',  '~');
         rc := WM.Propagate (win, ' ', WM.normalLow, 0, 0)  |
   36C : WriteLn (win)
  ELSE
    s[0] := ch;
    s[1] := 0C;
    rc := WM.Write (win, s)
  END
END Write;


PROCEDURE OpenTextWindow (VAR win: Window;
            xb, yb, xpixels, ypixels: CARDINAL; text: ARRAY OF CHAR);

VAR srows, scols, brow, bcol, wrows, wcols  : CARDINAL;

BEGIN                                 (* convert window parameters *)
  scols := (xpixels DIV 9) - 2;
  wcols := scols;
  wrows := (ypixels DIV 38);
  IF  wrows > 2  THEN  wrows := wrows-2  END;
  brow := (928 - yb - ypixels) DIV 38 + 1;
  bcol := xb DIV 9 + 1;
  CASE  text [0]  OF                  (* choose appropriate screensize *)
   'P' :  CASE text [t] OF
            'R' : srows := 80  |
            'A' : srows := 255
        END              |
   'C' : srows := 200    |
   'R' : srows := 100    |
   'D' : srows := 10
  END;

  rc := WM.CreateScreenWithWindow (win, srows, scols, brow, bcol, wrows,
```

```
                                    wcols);     (* create window *)
  rc := WM.SetBorderType (win, WM.singleLine);
  rc := WM.SetBorderAttribute (win, WM.normalLow);
  rc := WM.SetCursorType (win. WM.underlineCursor);
  rc := WM.UpdateOn ();
  rc := WM.LineWrapOn (win);
  rc := WM.ScreenWrapOn (win);
  rc := WM.SetDefaultAttribute (win, WM.normalHigh);
  WriteString (win, text);
  rc := WM.SetDefaultAttribute (win, WM.normalLow);
  IF  text[O] <> 'D'  THEN
    WriteLn (win)
  ELSE
    Write (win, ' ')
  END;
  rc := WM.OpenWindow (win)
END OpenTextWindow;


PR0CEDURE CloseTextWindow (win : Window);

VAR   ch, ch1                              : CHAR;
      rest, srows, scols, brow, bcol,
      crow, ccol, flags, pri               : CARDINAL;
      winN                                 : Window;
      s, rowup, rowdown, Home, End, Rest   : string;
      Helptext                             : STRING;

BEGIN
  (* define cursormovement keys (ANSI) *)
  rc := WM.GetScreenStatus (win, srows, scols, crow, ccol, flags);
  rest := 0;
  rowup := ' [1A';      rowdown := ' [1B';
  Home  := ' [0;0H';    End     := ' [;0H';
  rowup [0] := 33C;     rowdown [0] := 33C;     Rest := ' [B';
  Home [0]  := 33C;     End [0] := 33C;         Rest [0] := 33C;
  IF  srows > 127  THEN
    rest := srows - 127; srows := 127;
    ConvertCardinal (rest, 1, s);
    Insert (s, Rest, 2)
  END;
  ConvertCardinal (srows, 1, s);
  Insert (s, End, 2);
  rc := WM.ChangeWindow (win, 22, 78);          (* Enlarge window *)
  rc := WM.MoveWindow (win, 2, 1);
  rc := WM.WriteAt (win, 0, 0,  '~',  '~');
  rc := WM.OpenWindow (win);
  rc := WM.OpenWindow (winH);             (* Create or open helpwindow *)
  IF  rc # WM.OK  THEN
    rc := WM.CreateScreenWithWindow (winH, 1, 80, 0, 0, 1, 80);
    rc := WM.GetBorderType (winH, WM.noBox);
    rc := WM.SetDefaultAttribute (winH, WM.reverseLow);
    Concat ('KEYS:            Home, End,   and  : moving',
          '          ESC: Close window    ', Helptext);
    Helptext [27] := 30C;
    Helptext [33] := 31C;
    WriteString (winH. Helptext);
    rc := WM.OpenWindow (winH)
  END
```

```
  rc := WM.OpenWindow (win);
  REPEAT
    In0ut.Read (ch);
    IF  ch = 0C  THEN
      In0ut.Read (ch1);
      CASE  ch1  OF
        110C  : WriteString (win, rowup)  |
        107C  : WriteString (win, Home)   |
        117C  : WriteString (win, End);
                IF  rest <> 0 THEN
                  WriteString (win, Rest)
                END                       |
        120C  : WriteString (win, rowdown)
      END
    END
  UNTIL  ch = CHR (27);

  rc := WM.CloseWindow (win);
  rc := WM.CloseScreen (win);
  winN := 0;
  rc := WM.NextWindow (winN, pri);
  IF  winN = winH  THEN             (* Last window? ...           *)
    rc := WM.CloseWindow (winH);    (* then close helpwindow.     *)
    rc := WM.CloseScreen (winH);
    rc := WM.TerminateWM ()         (* and Terminate Window Machine *)
  END
END CloseTextWindow;


BEGIN
  IF  WM.InitWM (ADR (VBuf), SIZE (VBuf)) = WM.WMnotInitialized  THEN
                                          (* Window machine installed? *)
    InOut.WriteString ("WM not loaded; quit...");
    InOut.WriteLn;
    HALT
  END;
  winH := 255
END TextWindows.
```

In het programma General Rarser gebruikt Wirth een truc: hij gebruikt variabelen tijdelijk voor opslag van andere gegevens dan waarvoor de variabelen zijn gedeclareerd.
Deze wetenschap vergemakkelijkt wellicht het doorgronden van dit voorbeeld