AN AUTOMATIC FORMATTING PROGRAM FOR PASCAL

BY

Jon Hueras*

Henry Ledgard*

COINS Technical Report
TR76-14

AUGUST 1976

*University Computing Center and
Computer and Information Science Dept.
University of Massachusetts at Amherst
Amherst, MA 01002

AN AUTOMATIC FORMATTING PROGRAM FOR PASCAL

BY

Jon Hueras*

Henry Ledgard*

COINS Technical Report
TR76-14

AUGUST 1976

*University Computing Center and
Computer and Information Science Dept.
University of Massachusetts at Amherst
Amherst, MA 01002

PASCAL, like many languages, is a "free-format" language, in that there are no column-position or line-boundary restrictions on statements, declarations, or comments. Free-format languages have the advantage that the programmer may format his code in any way that well reflects the logical structuring of his program. Unfortunately, such languages also have the disadvantage that the programmer may just as easily impose no logical formatting scheme whatsoever. A program that is poorly formatted is often hard to maintain and modify, since its poor readability leads to confusion. In order to alleviate this problem, we have attempted to develop a set of useful formatting standards for PASCAL (see (1)). A program that has been thoughtfully formatted according to such a set of standards is said to be prettyprinted (see example below).

Imposing formatting restrictions necessarily imposes a burden on a programmer, particularly on a student programmer, since he must keypunch or type in the entire program himself. It is therefore useful to have a facility for taking arbitrarily formatted source code and automatically prettyprinting it. However, the design of any such prettyprinter must deal with several serious issues.

Typically, automatic prettyprinters take a heavy hand in formatting a program, right down to every last semicolon. Such a scheme either formats everything in a rigid fashion, which is bound to be displeasing, or else it provides the programmer with a voluminous set of "options". Furthermore, such a scheme must do a full syntax analysis on the program, which means that it falls prey to the bane of all compilers: error recovery. Thus, before a program may be prettyprinted, it must be completely written and debugged. If the programmer wishes to prettyprint a program still under development, he is out of luck, or else he must do it by hand, in which case he has no need for an automatic prettyprinter when he is done.

We believe that it is not necessary to impose more than a minimum set of restrictions, and that any prettyprinter should yield to the programmer's discretion beyond this minimum. No matter how many options a prettyprinter has,

it cannot possibly have one to please everyone in every possible case. We further believe that a prettyprinter should not commit itself to a full syntax analysis. It should only do prettyprinting on a local basis, dealing with individual constructs rather than entire programs.

In order to demonstrate these assertions, we have designed and implemented, in PASCAL, just such a prettyprinter. It is intended mostly as an editing aid, and thus does not include most of the "kitchen sink" facilities used by other prettyprinters. It simply rearranges the spacing and indentation of certain constructs in order to make the logical structure of a program more visually apparent. Furthermore, the prettyprinter forces only a minimum amount of spacing and indentation where needed. Any extra spaces or blank lines found in the program beyond the minimum required are left there. This leaves the programmer a good deal of flexibility to use as he sees fit.

The general strategy of the prettyprinter is simply to scan the program on a symbol by symbol basis, keeping track of the amount of space found between each symbol. If a distinguished symbol (such as "BEGIN", "UNTIL", etc.) is found, a table is consulted to see if any special prettyprinting actions are associated with that symbol. The most important actions are the indentation and de-indentation of the left margin, since they are the most difficult to handle. They are difficult because the prettyprinter needs more than local information to determine how and when to de-indent the margin. This is accomplished through the use of a stack. Each time the margin is indented, the symbol causing indentation is placed on the stack along with the previous position of the left margin. Thus, when de-indentation is called for, all that is needed is to pop the stack in order to determine what position the margin is to be restored to.

Since prettyprinting is done purely on a symbol by symbol basis, no syntax analysis is needed, and the program to be prettyprinted need not be syntactically correct. In fact, it need not even be a complete program. All that is required is that each individual construct be complete ("BEGIN"s must have their "END"s,

"REPEAT"s their "UNTIL"s, and so on). Furthermore, no error recovery is needed, since no errors are possible in the absence of a syntax analysis. An ill-defined construct may lead to ill-defined prettyprinting, but such errors are far from subtle and are easy to correct.

As an example of our prettyprinter's performance, consider the following program fragment:

```
TYPE SCALE=(CENTIGRADE,FAHRENHEIT);
FUNCTION CONVERT( DEGREES : INTEGER;
                  NEWSCALE: SCALE    ): INTEGER;
BEGIN IF NEWSCALE=CENTIGRADE THEN
CONVERT:=ROUND((9/5*DEGREES)+32)ELSE
CONVERT:=ROUND(5/9*(DEGREES-32))END;
```

When fed into our prettyprinter, the following output results:

```
TYPE SCALE = (CENTIGRADE,FAHRENHEIT);

FUNCTION CONVERT( DEGREES : INTEGER;
                  NEWSCALE: SCALE    ): INTEGER;
BEGIN
   IF NEWSCALE=CENTIGRADE
      THEN
         CONVERT := ROUND((9/5*DEGREES)+32)
      ELSE
         CONVERT := ROUND(5/9*(DEGREES-32))
END;
```

(Note that some care was taken to format the function header in a special way, and that the prettyprinter did not interfere.)

The prettyprinter that we have implemented is written entirely in standard PASCAL (2), and should compile and run using any PASCAL compiler. We have compiled it using the PASCAL 6000-3.4 compiler from Zurich, and run it using 11K (octal) of core on our CDC CYBER 74. The program, as written, is highly modularized and table-driven, and is therefore extremely easy to modify and upgrade. Our concern for proper abstraction leads to a certain amount of runtime overhead. Therefore, the program runs marginally slower than the PASCAL compiler. Execution speed could be increased considerably by making most of the variables global, and by eliminating some of the procedures and inserting them inline. However, we feel that the tradeoff is not worth it.

References:

(1) H. F. Ledgard, A. J. Singer, and J. F. Hueras: "A BASIS FOR EXECUTING PASCAL PROGRAMMERS" (to appear around December, 1976 in SIGPLAN notices).

(2) Kathleen Jensen and Nicklaus Wirth: Revised PASCAL Report (contained in PASCAL User Manual and Report, Springer-Verlag, New York).

```
 1
 2 (*=============================================================================*)
 3 (*                                                                             *)
 4 (*  PROGRAM TITLE: PASCAL PRETTYPRINTING PROGRAM                               *)
 5 (*                                                                             *)
 6 (*  AUTHORS: JON F. HUERAS AND HENRY F. LEDGARD                                *)
 7 (*           COMPUTER AND INFORMATION SCIENCE DEPARTMENT                        *)
 8 (*           UNIVERSITY OF MASSACHUSETTS, AMHERST                              *)
 9 (*           (EARLIER VERSIONS AND CONTRIBUTIONS BY RANDY CHOW                 *)
10 (*            AND JOHN GORMAN.)                                                *)
11 (*                                                                             *)
12 (*  PROGRAM SUMMARY:                                                           *)
13 (*                                                                             *)
14 (*     THIS PROGRAM TAKES AS INPUT A PASCAL PROGRAM AND                        *)
15 (*     REFORMATS THE PROGRAM ACCORDING TO A STANDARD SET OF                    *)
16 (*     PRETTYPRINTING RULES. THE PRETTYPRINTED PROGRAM IS GIVEN                *)
17 (*     AS OUTPUT.  THE PRETTYPRINTING RULES ARE GIVEN BELOW.                   *)
18 (*                                                                             *)
19 (*     AN IMPORTANT FEATURE IS THE PROVISION FOR THE USE OF EXTRA              *)
20 (*     SPACES AND EXTRA BLANK LINES.  THEY MAY BE FREELY INSERTED BY           *)
21 (*     THE USER IN ADDITION TO THE SPACES AND BLANK LINES INSERTED             *)
22 (*     BY THE PRETTYPRINTER.                                                   *)
23 (*                                                                             *)
24 (*     NO ATTEMPT IS MADE TO DETECT OR CORRECT SYNTACTIC ERRORS IN             *)
25 (*     THE USER'S PROGRAM.  HOWEVER, SYNTACTIC ERRORS MAY RESULT IN            *)
26 (*     ERRONEOUS PRETTYPRINTING.                                               *)
27 (*                                                                             *)
28 (*                                                                             *)
29 (*  INPUT FILE: INPUTFILE     - A FILE OF CHARACTERS, PRESUMABLY A             *)
30 (*                              PASCAL PROGRAM OR PROGRAM FRAGMENT.            *)
31 (*                                                                             *)
32 (*  OUTPUT FILES: OUTPUTFILE - THE PRETTYPRINTED PROGRAM.                      *)
33 (*                                                                             *)
34 (*                OUTPUT     - STANDARD PASCAL FILE FOR RUNTIME               *)
35 (*                             MESSAGES.                                       *)
36 (*                                                                             *)
37 (*                                                                             *)
38 (*=============================================================================*)
39
40
```

```
41
42 (*=============================================================================*)
43 (*                                                                             *)
44 (*                        PASCAL PRETTYPRINTING RULES                          *)
45 (*                                                                             *)
46 (*                                                                             *)
47 (*   [ GENERAL PRETTYPRINTING RULES ]                                          *)
48 (*                                                                             *)
49 (*    1.    ANY SPACES OR BLANK LINES BEYOND THOSE GENERATED BY THE            *)
50 (*      PRETTYPRINTER ARE LEFT ALONE.  THE USER IS ENCOURAGED, FOR THE         *)
51 (*      SAKE OF READABILITY, TO MAKE USE OF THIS FACILITY.                     *)
52 (*         IN ADDITION, COMMENTS ARE LEFT WHERE THEY ARE FOUND, UNLESS         *)
53 (*      THEY ARE SHIFTED RIGHT BY PRECEEDING TEXT ON A LINE.                   *)
54 (*                                                                             *)
55 (*    2.    ALL STATEMENTS AND DECLARATIONS BEGIN ON SEPARATE LINES.           *)
56 (*                                                                             *)
57 (*    3.    NO LINE MAY BE GREATER THAN 72 CHARACTERS LONG.  ANY LINE          *)
58 (*      LONGER THAN THIS IS CONTINUED ON A SEPARATE LINE.                      *)
59 (*                                                                             *)
60 (*    4.    THE KEYWORDS "BEGIN", "END", "REPEAT", AND "RECORD" ARE            *)
61 (*      FORCED TO STAND ON LINES BY THEMSELVES (OR POSSIBLY FOLLWED BY         *)
62 (*      SUPPORTING COMMENTS).                                                  *)
63 (*         IN  ADDITION, THE "UNTIL" CLAUSE OF A "REPEAT-UNTIL" STATE-         *)
64 (*      MENT IS FORCED TO START ON A NEW LINE.                                 *)
65 (*                                                                             *)
66 (*    5.    A BLANK LINE IS FORCED BEFORE THE KEYWORDS "PROGRAM",              *)
67 (*      "PROCEDURE", "FUNCTION", "LABEL", "CONST", "TYPE", AND "VAR".          *)
68 (*                                                                             *)
69 (*    6.    A SPACE IS FORCED BEFORE AND AFTER THE SYMBOLS ":=" AND            *)
70 (*      "=".  ADDITIONALLY, A SPACE IS FORCED AFTER THE SYMBOL ":".            *)
71 (*                                                                             *)
72 (*                                                                             *)
73 (*   [ INDENTATION RULES ]                                                     *)
74 (*                                                                             *)
75 (*    1.    THE BODIES OF "LABEL", "CONST", "TYPE", AND "VAR" DECLARA-         *)
76 (*      TIONS ARE INDENTED FROM THEIR CORRESPONDING DECLARATION HEADER         *)
77 (*      KEYWORDS.                                                              *)
78 (*                                                                             *)
79 (*    2.    THE BODIES OF "BEGIN-END", "REPEAT-UNTIL", "FOR", "WHILE",         *)
80 (*      "WITH", AND "CASE" STATEMENTS, AS WELL AS "RECORD-END" STRUC-          *)
81 (*      TURES AND "CASE" VARIANTS (TO ONE LEVEL) ARE INDENTED FROM             *)
82 (*      THEIR HEADER KEYWORDS.                                                 *)
83 (*                                                                             *)
84 (*    3.    AN "IF-THEN-ELSE" STATEMENT IS INDENTED AS FOLLOWS:                *)
85 (*                                                                             *)
86 (*              IF <EXPRESSION>                                                *)
87 (*                 THEN                                                        *)
88 (*                    <STATEMENT>                                              *)
89 (*                 ELSE                                                        *)
90 (*                    <STATEMENT>                                              *)
91 (*                                                                             *)
92 (*                                                                             *)
93 (*=============================================================================*)
94
95
```

```
 96
 97 (*=============================================================*)
 98 (*                                                             *)
 99 (*                      GENERAL ALGORITHM                      *)
100 (*                                                             *)
101 (*                                                             *)
102 (*     THE STRATEGY OF THE PRETTYPRINTER IS TO SCAN SYMBOLS FROM *)
103 (*   THE INPUT PROGRAM AND MAP EACH SYMBOL INTO A PRETTYPRINTING *)
104 (*   ACTION, INDEPENDENTLY OF THE CONTEXT IN WHICH THE SYMBOL    *)
105 (*   APPEARS.  THIS IS ACCOMPLISHED BY A TABLE OF PRETTYPRINTING *)
106 (*   OPTIONS.                                                   *)
107 (*                                                              *)
108 (*     FOR EACH DISTINGUISHED SYMBOL IN THE TABLE, THERE IS AN  *)
109 (*   ASSOCIATED SET OF OPTIONS.  IF THE OPTION HAS BEEN SELECTED FOR *)
110 (*   THE SYMBOL BEING SCANNED, THEN THE ACTION CORRESPONDING WITH *)
111 (*   EACH OPTION IS PERFORMED.                                  *)
112 (*                                                              *)
113 (*     THE BASIC ACTIONS INVOLVED IN PRETTYPRINTING ARE THE INDENT- *)
114 (*   ATION AND DE-INDENTATION OF THE MARGIN.  EACH TIME THE MARGIN IS *)
115 (*   INDENTED, THE PREVIOUS VALUE OF THE MARGIN IS PUSHED ONTO A *)
116 (*   STACK, ALONG WITH THE NAME OF THE SYMBOL THAT CAUSED IT TO BE *)
117 (*   INDENTED.  EACH TIME THE MARGIN IS DE-INDENTED, THE STACK IS *)
118 (*   POPPED OFF TO OBTAIN THE PREVIOUS VALUE OF THE MARGIN.     *)
119 (*                                                              *)
120 (*     THE PRETTYPRINTING OPTIONS ARE PROCESSED IN THE FOLLOWING *)
121 (*   ORDER, AND INVOKE THE FOLLOWING ACTIONS:                   *)
122 (*                                                              *)
123 (*                                                              *)
124 (*     CRSUPPRESS       - IF A CARRIAGE RETURN HAS BEEN INSERTED *)
125 (*                        FOLLOWING THE PREVIOUS SYMBOL, THEN IT IS *)
126 (*                        INHIBITED UNTIL THE NEXT SYMBOL IS PRINTED. *)
127 (*                                                              *)
128 (*     CRBEFORE         - A CARRIAGE RETURN IS INSERTED BEFORE THE *)
129 (*                        CURRENT SYMBOL (UNLESS ONE IS ALREADY THERE) *)
130 (*                                                              *)
131 (*     BLANKLINEBEFORE - A BLANK LINE IS INSERTED BEFORE THE CURRENT *)
132 (*                        SYMBOL (UNLESS ALREADY THERE).        *)
133 (*                                                              *)
134 (*     DINDENTONKEYS    - IF ANY OF THE SPECIFIED KEYS ARE ON TOP OF *)
135 (*                        OF THE STACK, THE STACK IS POPPED, DE-INDEN- *)
136 (*                        TING THE MARGIN.  THE PROCESS IS REPEATED *)
137 (*                        UNTIL THE TOP OF THE STACK IS NOT ONE OF THE *)
138 (*                        SPECIFIED KEYS.                       *)
139 (*                                                              *)
140 (*     DINDENT          - THE STACK IS UNCONDITIONALLY POPPED AND THE *)
141 (*                        MARGIN IS DE-INDENTED.                *)
142 (*                                                              *)
143 (*     SPACEBEFORE      - A SPACE IS INSERTED BEFORE THE SYMBOL BEING *)
144 (*                        SCANNED (UNLESS ALREADY THERE).       *)
145 (*                                                              *)
146 (*     [ THE SYMBOL IS PRINTED AT THIS POINT ]                  *)
147 (*                                                              *)
148 (*     SPACEAFTER       - A SPACE IS INSERTED AFTER THE SYMBOL BEING *)
149 (*                        SCANNED (UNLESS ALREADY THERE).       *)
150 (*                                                              *)
151 (*     GOBBLESYMBOLS    - SYMBOLS ARE CONTINUOUSLY SCANNED AND PRINTED *)
152 (*                        WITHOUT ANY PROCESSING UNTIL ONE OF THE *)
153 (*                        SPECIFIED SYMBOLS IS SEEN (BUT NOT GOBBLED). *)
154 (*                                                              *)
155 (*     INDENTBYTAB      - THE MARGIN IS INDENTED BY A STANDARD AMOUNT *)
156 (*                        FROM THE PREVIOUS MARGIN.             *)
157 (*                                                              *)
158 (*     INDENTTOCLP      - THE MARGIN IS INDENTED TO THE CURRENT LINE *)
159 (*                        POSITION.                             *)
160 (*                                                              *)
161 (*     CRAFTER          - A CARRIAGE RETURN IS INSERTED FOLLOWING THE *)
162 (*                        SYMBOL SCANNED.                       *)
163 (*                                                              *)
164 (*                                                              *)
165 (*                                                              *)
166 (*=============================================================*)
167
168
```

```
169
170 PROGRAM PRETTYPRINT( (* FROM *)  INPUTFILE,
171                      (* TO *)    OUTPUTFILE,
172                      (* USING *) OUTPUT     );
173
174
175 CONST
176
177       MAXSYMBOLSIZE = 200; (* THE MAXIMUM SIZE (IN CHARACTERS) OF A   *)
178                            (* SYMBOL SCANNED BY THE LEXICAL SCANNER.  *)
179
180       MAXSTACKSIZE  = 100; (* THE MAXIMUM NUMBER OF SYMBOLS CAUSING   *)
181                            (* INDENTATION THAT MAY BE STACKED.        *)
182
183       MAXKEYLENGTH  =  10; (* THE MAXIMUM LENGTH (IN CHARACTERS) OF A *)
184                            (* PASCAL RESERVED KEYWORD.                *)
185       MAXLINESIZE   =  72; (* THE MAXIMUM SIZE (IN CHARACTERS) OF A   *)
186                            (* LINE OUTPUT BY THE PRETTYPRINTER.       *)
187
188       SLOWFAIL1     =  30; (* UP TO THIS COLUMN POSITION, EACH TIME   *)
189                            (* "INDENTBYTAB" IS INVOKED, THE MARGIN    *)
190                            (* WILL BE INDENTED BY "INDENT1".          *)
191
192       SLOWFAIL2     =  48; (* UP TO THIS COLUMN POSITION, EACH TIME   *)
193                            (* "INDENTBYTAB" IS INVOKED, THE MARGIN    *)
194                            (* WILL BE INDENTED BY "INDENT2".  BEYOND  *)
195                            (* THIS, NO INDENTATION OCCURS.            *)
196
197       INDENT1       =   3;
198
199       INDENT2       =   1;
200
201
202       SPACE = ' ';
203
204
```

```
205
206 TYPE
207
208       KEYSYMBOL = ( PROGSYM,     FUNCSYM,      PROCSYM,
209                     LABELSYM,    CONSTSYM,     TYPESYM,    VARSYM,
210                     BEGINSYM,    REPEATSYM,    RECORDSYM,
211                     CASESYM,     CASEVARSYM,   OFSYM,
212                     FORSYM,      WHILESYM,     WITHSYM,    DOSYM,
213                     IFSYM,       THENSYM,      ELSESYM,
214                     ENDSYM,      UNTILSYM,
215                     BECOMES,     OPENCOMMENT,  CLOSECOMMENT,
216                     SEMICOLON,   COLON,        EQUALS,
217                     OPENPAREN,   CLOSEPAREN,   PERIOD,
218                     ENDOFFILE,
219                     OTHERSYM );
220
221       OPTION = ( CRSUPPRESS,
222                  CRBEFORE,
223                  BLANKLINEBEFORE,
224                  DINDENTONKEYS,
225                  DINDENT,
226                  SPACEBEFORE,
227                  SPACEAFTER,
228                  GOBBLESYMBOLS,
229                  INDENTBYTAB,
230                  INDENTTOCLP,
231                  CRAFTER );
232
233       OPTIONSET = SET OF OPTION;
234
235       KEYSYMSET = SET OF KEYSYMBOL;
236
237       TABLEENTRY = RECORD
238                       OPTIONSSELECTED   : OPTIONSET;
239                       DINDENTSYMBOLS    : KEYSYMSET;
240                       GOBBLETERMINATORS: KEYSYMSET
241                    END;
242
243       OPTIONTABLE = ARRAY [ KEYSYMBOL ] OF TABLEENTRY;
244
245
```

```
246
247       KEY = PACKED ARRAY [ 1..MAXKEYLENGTH ] OF CHAR;
248
249
250       KEYWORDTABLE = ARRAY [ PROGSYM..UNTILSYM ] OF KEY;
251
252
253       SPECIALCHAR = PACKED ARRAY [ 1..2 ] OF CHAR;
254
255       DBLCHRSET = SET OF BECOMES..OPENCOMMENT;
256
257       DBLCHARTABLE = ARRAY [ BECOMES..OPENCOMMENT ] OF SPECIALCHAR;
258
259       SGLCHARTABLE = ARRAY [ SEMICOLON..PERIOD ] OF CHAR;
260
261
262       STRING = ARRAY [ 1..MAXSYMBOLSIZE ] OF CHAR;
263
264       SYMBOL = RECORD
265                   NAME          : KEYSYMBOL;
266                   VALUE         : STRING;
267                   LENGTH        : INTEGER;
268                   SPACESBEFORE: INTEGER;
269                   CRSBEFORE     : INTEGER
270                END;
271
272       SYMBOLINFO = ^SYMBOL;
273
274
275       CHARNAME = ( LETTER,     DIGIT,     BLANK,     QUOTE,
276                    ENDOFLINE, FILEMARK, OTHERCHAR        );
277
278       CHARINFO = RECORD
279                     NAME : CHARNAME;
280                     VALUE: CHAR
281                  END;
282
283
284       STACKENTRY = RECORD
285                       INDENTSYMBOL: KEYSYMBOL;
286                       PREVMARGIN  : INTEGER
287                    END;
288
289       SYMBOLSTACK = ARRAY [ 1..MAXSTACKSIZE ] OF STACKENTRY;
290
291
```

```
292
293 VAR
294
295     INPUTFILE,
296     OUTPUTFILE: TEXT;
297
298     RECORDSEEN: BOOLEAN;
299
300     CURRCHAR,
301     NEXTCHAR: CHARINFO;
302
303     CURRSYM,
304     NEXTSYM: SYMBOLINFO;
305
306     CRPENDING: BOOLEAN;
307
308     PPOPTION: OPTIONTABLE;
309
310     KEYWORD: KEYWORDTABLE;
311
312     DBLCHARS: DBLCHRSET;
313
314     DBLCHAR: DBLCHARTABLE;
315     SGLCHAR: SGLCHARTABLE;
316
317     STACK: SYMBOLSTACK;
318     TOP  : INTEGER;
319
320     CURRLINEPOS,
321     CURRMARGIN :  INTEGER;
322
323
```

```
324
325 PROCEDURE GETCHAR( (* FROM *)       VAR INPUTFILE : TEXT;
326                    (* UPDATING *)   VAR NEXTCHAR  : CHARINFO;
327                    (* RETURNING *) VAR CURRCHAR  : CHARINFO );
328
329 BEGIN (* GETCHAR *)
330
331    CURRCHAR := NEXTCHAR;
332
333    WITH NEXTCHAR DO
334       BEGIN
335
336          IF EOF(INPUTFILE)
337             THEN
338                NAME  := FILEMARK
339
340     ELSE IF EOLN(INPUTFILE)
341             THEN
342                NAME  := ENDOFLINE
343
344     ELSE IF INPUTFILE^ IN ['A'..'Z']
345             THEN
346                NAME  := LETTER
347
348     ELSE IF INPUTFILE^ IN ['0'..'9']
349             THEN
350                NAME  := DIGIT
351
352     ELSE IF INPUTFILE^ = ''''
353             THEN
354                NAME  := QUOTE
355
356     ELSE IF INPUTFILE^ = SPACE
357             THEN
358                NAME  := BLANK
359
360     ELSE NAME := OTHERCHAR;
361
362
363          IF NAME IN [ FILEMARK, ENDOFLINE ]
364             THEN
365                VALUE := SPACE
366             ELSE
367                VALUE := INPUTFILE^;
368
369          IF NAME <> FILEMARK
370             THEN
371                GET(INPUTFILE)
372
373       END (* WITH *)
374
375 END; (* GETCHAR *)
376
377
```

```
378
379 PROCEDURE STORENEXTCHAR( (* FROM *)         VAR INPUTFILE : TEXT;
380                          (* UPDATING *)     VAR LENGTH    : INTEGER;
381                                              VAR CURRCHAR,
382                                                  NEXTCHAR  : CHARINFO;
383                          (* PLACING IN *)   VAR VALUE     : STRING   );
384
385 BEGIN (* STORENEXTCHAR *)
386
387    GETCHAR( (* FROM *)       INPUTFILE,
388             (* UPDATING *)   NEXTCHAR,
389             (* RETURNING *) CURRCHAR  );
390
391    IF LENGTH < MAXSYMBOLSIZE
392       THEN
393          BEGIN
394
395             LENGTH := LENGTH + 1;
396
397             VALUE [LENGTH] := CURRCHAR.VALUE
398
399          END
400
401 END; (* STORENEXTCHAR *)
402
403
```

```
404
405 PROCEDURE SKIPSPACES( (* IN *)         VAR INPUTFILE     : TEXT;
406                       (* UPDATING *)   VAR CURRCHAR,
407                                             NEXTCHAR      : CHARINFO;
408                       (* RETURNING *) VAR SPACESBEFORE,
409                                             CRSBEFORE     : INTEGER  );
410
411 BEGIN (* SKIPSPACES *)
412
413    SPACESBEFORE := 0;
414    CRSBEFORE    := 0;
415
416    WHILE NEXTCHAR.NAME IN [ BLANK, ENDOFLINE ] DO
417       BEGIN
418
419          GETCHAR( (* FROM *)       INPUTFILE,
420                   (* UPDATING *)   NEXTCHAR,
421                   (* RETURNING *) CURRCHAR  );
422
423          CASE CURRCHAR.NAME OF
424
425             BLANK     : SPACESBEFORE := SPACESBEFORE + 1;
426
427             ENDOFLINE : BEGIN
428                            CRSBEFORE    := CRSBEFORE + 1;
429                            SPACESBEFORE := 0
430                         END
431
432          END (* CASE *)
433
434       END (* WHILE *)
435
436 END; (* SKIPSPACES *)
437
438
```

```
439
440 PROCEDURE GETCOMMENT( (* FROM *)      VAR INPUTFILE : TEXT;
441                       (* UPDATING *) VAR CURRCHAR,
442                                           NEXTCHAR  : CHARINFO;
443                                       VAR NAME      : KEYSYMBOL;
444                                       VAR VALUE     : STRING;
445                                       VAR LENGTH    : INTEGER   );
446
447 BEGIN (* GETCOMMENT *)
448
449    NAME := OPENCOMMENT;
450
451    WHILE NOT(     ((CURRCHAR.VALUE = '*') AND (NEXTCHAR.VALUE = ')'))
452                OR (NEXTCHAR.NAME = ENDOFLINE)
453                OR (NEXTCHAR.NAME = FILEMARK)) DO
454
455       STORENEXTCHAR( (* FROM *)      INPUTFILE,
456                      (* UPDATING *) LENGTH,
457                                     CURRCHAR,
458                                     NEXTCHAR,
459                      (* IN *)       VALUE     );
460
461
462    IF (CURRCHAR.VALUE = '*') AND (NEXTCHAR.VALUE = ')')
463       THEN
464          BEGIN
465
466             STORENEXTCHAR( (* FROM *)      INPUTFILE,
467                            (* UPDATING *) LENGTH,
468                                           CURRCHAR,
469                                           NEXTCHAR,
470                            (* IN *)       VALUE     );
471
472             NAME := CLOSECOMMENT
473
474          END
475
476 END; (* GETCOMMENT *)
477
478
```

```
479
480 FUNCTION IDTYPE( (* OF *)          VALUE  : STRING;
481                  (* USING *)       LENGTH : INTEGER )
482                  (* RETURNING *)                      : KEYSYMBOL;
483
484 VAR
485     I: INTEGER;
486
487     KEYVALUE: KEY;
488
489     HIT: BOOLEAN;
490
491     THISKEY: KEYSYMBOL;
492
493
494 BEGIN (* IDTYPE *)
495
496    IDTYPE := OTHERSYM;
497
498    IF LENGTH <= MAXKEYLENGTH
499       THEN
500          BEGIN
501
502             FOR I := 1 TO LENGTH DO
503                KEYVALUE [I] := VALUE [I];
504
505             FOR I := LENGTH+1 TO MAXKEYLENGTH DO
506                KEYVALUE [I] := SPACE;
507
508             THISKEY := PROGSYM;
509             HIT     := FALSE;
510
511             WHILE NOT(HIT OR (PRED(THISKEY) = UNTILSYM)) DO
512                IF KEYVALUE = KEYWORD [THISKEY]
513                   THEN
514                      HIT := TRUE
515                   ELSE
516                      THISKEY := SUCC(THISKEY);
517
518             IF HIT
519                THEN
520                   IDTYPE := THISKEY
521
522          END;
523
524 END; (* IDTYPE *)
525
526
```

```
527
528 PROCEDURE GETIDENTIFIER( (* FROM *)       VAR INPUTFILE : TEXT;
529                          (* UPDATING *)   VAR CURRCHAR,
530                                               NEXTCHAR  : CHARINFO;
531                          (* RETURNING *)  VAR NAME      : KEYSYMBOL;
532                                           VAR VALUE     : STRING;
533                                           VAR LENGTH    : INTEGER   );
534
535 BEGIN (* GETIDENTIFIER *)
536
537    WHILE NEXTCHAR.NAME IN [ LETTER, DIGIT ] DO
538
539       STORENEXTCHAR( (* FROM *)     INPUTFILE,
540                      (* UPDATING *) LENGTH,
541                                     CURRCHAR,
542                                     NEXTCHAR,
543                      (* IN *)       VALUE     );
544
545
546    NAME := IDTYPE( (* OF *)    VALUE,
547                    (* USING *) LENGTH );
548
549    IF NAME IN [ RECORDSYM, CASESYM, ENDSYM ]
550       THEN
551          CASE NAME OF
552
553             RECORDSYM : RECORDSEEN := TRUE;
554
555             CASESYM   : IF RECORDSEEN
556                            THEN
557                               NAME := CASEVARSYM;
558
559             ENDSYM    : RECORDSEEN := FALSE
560
561          END (* CASE *)
562
563 END; (* GETIDENTIFIER *)
564
565
```

```
566
567 PROCEDURE GETNUMBER( (* FROM *)       VAR INPUTFILE : TEXT;
568                      (* UPDATING *)   VAR CURRCHAR,
569                                           NEXTCHAR  : CHARINFO;
570                      (* RETURNING *)  VAR NAME      : KEYSYMBOL;
571                                       VAR VALUE     : STRING;
572                                       VAR LENGTH    : INTEGER   );
573
574 BEGIN (* GETNUMBER *)
575
576    WHILE NEXTCHAR.NAME = DIGIT DO
577
578       STORENEXTCHAR( (* FROM *)      INPUTFILE,
579                      (* UPDATING *)  LENGTH,
580                                      CURRCHAR,
581                                      NEXTCHAR,
582                      (* IN *)        VALUE     );
583
584
585    NAME := OTHERSYM
586
587 END; (* GETNUMBER *)
588
589
```

```
590
591 PROCEDURE GETCHARLITERAL( (* FROM *)       VAR INPUTFILE : TEXT;
592                           (* UPDATING *)   VAR CURRCHAR,
593                                                NEXTCHAR  : CHARINFO;
594                           (* RETURNING *)  VAR NAME      : KEYSYMBOL;
595                                            VAR VALUE     : STRING;
596                                            VAR LENGTH    : INTEGER   );
597
598 BEGIN (* GETCHARLITERAL *)
599
600    WHILE NEXTCHAR.NAME = QUOTE DO
601       BEGIN
602
603          STORENEXTCHAR( (* FROM *)      INPUTFILE,
604                         (* UPDATING *) LENGTH,
605                                        CURRCHAR,
606                                        NEXTCHAR,
607                         (* IN *)       VALUE     );
608
609          WHILE NOT(NEXTCHAR.NAME IN [ QUOTE, ENDOFLINE, FILEMARK ]) DO
610
611             STORENEXTCHAR( (* FROM *)      INPUTFILE,
612                            (* UPDATING *) LENGTH,
613                                           CURRCHAR,
614                                           NEXTCHAR,
615                            (* IN *)       VALUE     );
616
617
618          IF NEXTCHAR.NAME = QUOTE
619             THEN
620                STORENEXTCHAR( (* FROM *)      INPUTFILE,
621                               (* UPDATING *) LENGTH,
622                                              CURRCHAR,
623                                              NEXTCHAR,
624                               (* IN *)       VALUE     )
625
626       END;
627
628
629    NAME := OTHERSYM
630
631 END; (* GETCHARLITERAL *)
632
633
```

```
634
635 FUNCTION CHARTYPE( (* OF *)          CURRCHAR,
636                                       NEXTCHAR : CHARINFO )
637                     (* RETURNING *)                          : KEYSYMBOL;
638
639 VAR
640     NEXTTWOCHARS: SPECIALCHAR;
641
642     HIT: BOOLEAN;
643
644     THISCHAR: KEYSYMBOL;
645
646
647 BEGIN (* CHARTYPE *)
648
649    NEXTTWOCHARS[1] := CURRCHAR.VALUE;
650    NEXTTWOCHARS[2] := NEXTCHAR.VALUE;
651
652    THISCHAR := BECOMES;
653    HIT      := FALSE;
654
655    WHILE NOT(HIT OR (THISCHAR = CLOSECOMMENT)) DO
656       IF NEXTTWOCHARS = DBLCHAR [THISCHAR]
657          THEN
658             HIT := TRUE
659          ELSE
660             THISCHAR := SUCC(THISCHAR);
661
662    IF NOT HIT
663      THEN
664         BEGIN
665
666            THISCHAR := SEMICOLON;
667
668            WHILE NOT(HIT OR (PRED(THISCHAR) = PERIOD)) DO
669               IF CURRCHAR.VALUE = SGLCHAR [THISCHAR]
670                  THEN
671                     HIT := TRUE
672                  ELSE
673                     THISCHAR := SUCC(THISCHAR)
674
675         END;
676
677    IF HIT
678       THEN
679          CHARTYPE := THISCHAR
680       ELSE
681          CHARTYPE := OTHERSYM
682
683 END; (* CHARTYPE *)
684
685
```

```
686
687 PROCEDURE GETSPECIALCHAR( (* FROM *)       VAR INPUTFILE : TEXT;
688                           (* UPDATING *)   VAR CURRCHAR,
689                                                NEXTCHAR  : CHARINFO;
690                           (* RETURNING *)  VAR NAME      : KEYSYMBOL;
691                                            VAR VALUE     : STRING;
692                                            VAR LENGTH    : INTEGER   );
693
694 BEGIN (* GETSPECIALCHAR *)
695
696    STORENEXTCHAR( (* FROM *)      INPUTFILE,
697                   (* UPDATING *) LENGTH,
698                                  CURRCHAR,
699                                  NEXTCHAR,
700                   (* IN *)       VALUE     );
701
702    NAME := CHARTYPE( (* OF *) CURRCHAR,
703                                 NEXTCHAR );
704
705    IF NAME IN DBLCHARS
706       THEN
707
708          STORENEXTCHAR( (* FROM *)      INPUTFILE,
709                         (* UPDATING *) LENGTH,
710                                        CURRCHAR,
711                                        NEXTCHAR,
712                         (* IN *)       VALUE     )
713
714 END; (* GETSPECIALCHAR *)
715
716
```

```
717
718 PROCEDURE GETNEXTSYMBOL( (* FROM *)      VAR INPUTFILE : TEXT;
719                          (* UPDATING *)  VAR CURRCHAR,
720                                               NEXTCHAR  : CHARINFO;
721                          (* RETURNING *) VAR NAME      : KEYSYMBOL;
722                                          VAR VALUE     : STRING;
723                                          VAR LENGTH    : INTEGER   );
724
725 BEGIN (* GETNEXTSYMBOL *)
726
727    CASE NEXTCHAR.NAME OF
728
729       LETTER      : GETIDENTIFIER( (* FROM *)      INPUTFILE,
730                                    (* UPDATING *)  CURRCHAR,
731                                                    NEXTCHAR,
732                                    (* RETURNING *) NAME,
733                                                    VALUE,
734                                                    LENGTH    );
735
736       DIGIT       : GETNUMBER( (* FROM *)      INPUTFILE,
737                                (* UPDATING *)  CURRCHAR,
738                                                NEXTCHAR,
739                                (* RETURNING *) NAME,
740                                                VALUE,
741                                                LENGTH    );
742
743       QUOTE       : GETCHARLITERAL( (* FROM *)      INPUTFILE,
744                                     (* UPDATING *)  CURRCHAR,
745                                                     NEXTCHAR,
746                                     (* RETURNING *) NAME,
747                                                     VALUE,
748                                                     LENGTH    );
749
750       OTHERCHAR   : BEGIN
751
752                        GETSPECIALCHAR( (* FROM *)      INPUTFILE,
753                                        (* UPDATING *)  CURRCHAR,
754                                                        NEXTCHAR,
755                                        (* RETURNING *) NAME,
756                                                        VALUE,
757                                                        LENGTH    );
758
759                        IF NAME = OPENCOMMENT
760                           THEN
761                              GETCOMMENT( (* FROM *)      INPUTFILE,
762                                          (* UPDATING *) CURRCHAR,
763                                                         NEXTCHAR,
764                                                         NAME,
765                                                         VALUE,
766                                                         LENGTH    )
767
768                        END;
769
770       FILEMARK    : NAME := ENDOFFILE
771
772    END (* CASE *)
773
774 END; (* GETNEXTSYMBOL *)
775
776
```

```
777
778 PROCEDURE GETSYMBOL( (* FROM *)       VAR INPUTFILE : TEXT;
779                       (* UPDATING *)   VAR NEXTSYM   : SYMBOLINFO;
780                       (* RETURNING *) VAR CURRSYM   : SYMBOLINFO );
781
782 VAR
783     DUMMY: SYMBOLINFO;
784
785
786 BEGIN (* GETSYMBOL *)
787
788    DUMMY   := CURRSYM;
789    CURRSYM := NEXTSYM;
790    NEXTSYM := DUMMY  ;
791
792    WITH NEXTSYM^ DO
793       BEGIN
794
795          SKIPSPACES( (* IN *)          INPUTFILE,
796                      (* UPDATING *)    CURRCHAR,
797                                        NEXTCHAR,
798                      (* RETURNING *)   SPACESBEFORE,
799                                        CRSBEFORE     );
800          LENGTH := 0;
801
802          IF CURRSYM^.NAME = OPENCOMMENT
803             THEN
804                GETCOMMENT( (* FROM *)       INPUTFILE,
805                            (* UPDATING *)   CURRCHAR,
806                                             NEXTCHAR,
807                            (* RETURNING *)  NAME,
808                                             VALUE,
809                                             LENGTH    )
810             ELSE
811                GETNEXTSYMBOL( (* FROM *)       INPUTFILE,
812                               (* UPDATING *)   CURRCHAR,
813                                                NEXTCHAR,
814                               (* RETURNING *)  NAME,
815                                                VALUE,
816                                                LENGTH    )
817
818       END (* WITH *)
819
820 END; (* GETSYMBOL *)
821
822
```

```
823
824 PROCEDURE INITIALIZE( (* RETURNING *)
825
826                          VAR INPUTFILE,
827                              OUTPUTFILE   : TEXT;
828
829                          VAR TOPOFSTACK   : INTEGER;
830
831                          VAR CURRLINEPOS,
832                              CURRMARGIN   : INTEGER;
833
834                          VAR KEYWORD      : KEYWORDTABLE;
835
836                          VAR DBLCHARS     : DBLCHRSET;
837
838                          VAR DBLCHAR      : DBLCHARTABLE;
839
840                          VAR SGLCHAR      : SGLCHARTABLE;
841
842                          VAR RECORDSEEN   : BOOLEAN;
843
844                          VAR CURRCHAR,
845                              NEXTCHAR     : CHARINFO;
846
847                          VAR CURRSYM,
848                              NEXTSYM      : SYMBOLINFO;
849
850                          VAR PPOPTION     : OPTIONTABLE    );
851
852
```

```
853
854 BEGIN (* INITIALIZE *)
855
856     RESET(INPUTFILE);
857     REWRITE(OUTPUTFILE);
858
859     TOPOFSTACK  := 0;
860     CURRLINEPOS := 0;
861     CURRMARGIN  := 0;
862
863
864     KEYWORD [ PROGSYM     ] := 'PROGRAM   ' ;
865     KEYWORD [ FUNCSYM     ] := 'FUNCTION  ' ;
866     KEYWORD [ PROCSYM     ] := 'PROCEDURE ' ;
867     KEYWORD [ LABELSYM    ] := 'LABEL     ' ;
868     KEYWORD [ CONSTSYM    ] := 'CONST     ' ;
869     KEYWORD [ TYPESYM     ] := 'TYPE      ' ;
870     KEYWORD [ VARSYM      ] := 'VAR       ' ;
871     KEYWORD [ BEGINSYM    ] := 'BEGIN     ' ;
872     KEYWORD [ REPEATSYM   ] := 'REPEAT    ' ;
873     KEYWORD [ RECORDSYM   ] := 'RECORD    ' ;
874     KEYWORD [ CASESYM     ] := 'CASE      ' ;
875     KEYWORD [ CASEVARSYM  ] := 'CASE      ' ;
876     KEYWORD [ OFSYM       ] := 'OF        ' ;
877     KEYWORD [ FORSYM      ] := 'FOR       ' ;
878     KEYWORD [ WHILESYM    ] := 'WHILE     ' ;
879     KEYWORD [ WITHSYM     ] := 'WITH      ' ;
880     KEYWORD [ DOSYM       ] := 'DO        ' ;
881     KEYWORD [ IFSYM       ] := 'IF        ' ;
882     KEYWORD [ THENSYM     ] := 'THEN      ' ;
883     KEYWORD [ ELSESYM     ] := 'ELSE      ' ;
884     KEYWORD [ ENDSYM      ] := 'END       ' ;
885     KEYWORD [ UNTILSYM    ] := 'UNTIL     ' ;
886
887
888     DBLCHARS := [ BECOMES, OPENCOMMENT ];
889
890     DBLCHAR [ BECOMES     ]  := ':=' ;
891     DBLCHAR [ OPENCOMMENT ]  := '(*' ;
892
893     SGLCHAR [ SEMICOLON   ]  := ';' ;
894     SGLCHAR [ COLON       ]  := ':' ;
895     SGLCHAR [ EQUALS      ]  := '=' ;
896     SGLCHAR [ OPENPAREN   ]  := '(' ;
897     SGLCHAR [ CLOSEPAREN  ]  := ')' ;
898     SGLCHAR [ PERIOD      ]  := '.' ;
899
900     RECORDSEEN := FALSE;
901
902
903     GETCHAR( (* FROM *)       INPUTFILE,
904              (* UPDATING *)   NEXTCHAR,
905              (* RETURNING *) CURRCHAR  );
906
907     NEW(CURRSYM);
908     NEW(NEXTSYM);
909
910     GETSYMBOL( (* FROM *)       INPUTFILE,
911                (* UPDATING *)   NEXTSYM,
912                (* RETURNING *) CURRSYM  );
913
```

```
915
916     WITH PPOPTION [ PROGSYM ] DO
917        BEGIN
918           OPTIONSSELECTED    := [ BLANKLINEBEFORE,
919                                   SPACEAFTER ];
920
921           DINDENTSYMBOLS     := [];
922           GOBBLETERMINATORS  := []
923        END;
924
925     WITH PPOPTION [ FUNCSYM ] DO
926        BEGIN
927           OPTIONSSELECTED    := [ BLANKLINEBEFORE,
928                                   DINDENTONKEYS,
929                                   SPACEAFTER ];
930
931           DINDENTSYMBOLS     := [ LABELSYM,
932                                   CONSTSYM,
933                                   TYPESYM,
934                                   VARSYM ];
935
936           GOBBLETERMINATORS  := []
937        END;
938
939     WITH PPOPTION [ PROCSYM ] DO
940        BEGIN
941           OPTIONSSELECTED    := [ BLANKLINEBEFORE,
942                                   DINDENTONKEYS,
943                                   SPACEAFTER ];
944
945           DINDENTSYMBOLS     := [ LABELSYM,
946                                   CONSTSYM,
947                                   TYPESYM,
948                                   VARSYM ];
949
950           GOBBLETERMINATORS  := []
951        END;
952
953     WITH PPOPTION [ LABELSYM ] DO
954        BEGIN
955           OPTIONSSELECTED    := [ CRBEFORE,
956                                   SPACEAFTER,
957                                   INDENTTOCLP ];
958
959           DINDENTSYMBOLS     := [];
960           GOBBLETERMINATORS  := []
961        END;
962
963     WITH PPOPTION [ CONSTSYM ] DO
964        BEGIN
965           OPTIONSSELECTED    := [ CRBEFORE,
966                                   DINDENTONKEYS,
967                                   SPACEAFTER,
968                                   INDENTTOCLP ];
969
970           DINDENTSYMBOLS     := [ LABELSYM ];
971           GOBBLETERMINATORS  := []
972        END;
973
974     WITH PPOPTION [ TYPESYM ] DO
975        BEGIN
976           OPTIONSSELECTED    := [ CRBEFORE,
977                                   DINDENTONKEYS,
978                                   SPACEAFTER,
979                                   INDENTTOCLP ];
980
981           DINDENTSYMBOLS     := [ LABELSYM,
982                                   CONSTSYM ];
983
984           GOBBLETERMINATORS  := []
985        END;
986
987     WITH PPOPTION [ VARSYM ] DO
988        BEGIN
989           OPTIONSSELECTED    := [ CRBEFORE,
990                                   DINDENTONKEYS,
991                                   SPACEAFTER,
992                                   INDENTTOCLP ];
993
994           DINDENTSYMBOLS     := [ LABELSYM,
995                                   CONSTSYM,
996                                   TYPESYM ];
997
998           GOBBLETERMINATORS  := []
999        END;
1000
1001    WITH PPOPTION [ BEGINSYM ] DO
1002       BEGIN
1003          OPTIONSSELECTED    := [ DINDENTONKEYS,
1004                                  INDENTBYTAB,
1005                                  CRAFTER ];
1006
1007          DINDENTSYMBOLS     := [ LABELSYM,
1008                                  CONSTSYM,
1009                                  TYPESYM,
1010                                  VARSYM ];
1011
1012          GOBBLETERMINATORS  := []
1013       END;
```

```
1001
1002    WITH PPOPTION [ REPEATSYM ] DO
1003       BEGIN
1004          OPTIONSSELECTED   := [ INDENTBYTAB,
1005                                 CRAFTER ];
1006          DINDENTSYMBOLS    := [];
1007          GOBBLETERMINATORS := []
1008       END;
1009
1010    WITH PPOPTION [ RECORDSYM ] DO
1011       BEGIN
1012          OPTIONSSELECTED   := [ INDENTBYTAB,
1013                                 CRAFTER ];
1014          DINDENTSYMBOLS    := [];
1015          GOBBLETERMINATORS := []
1016       END;
1017
1018    WITH PPOPTION [ CASESYM ] DO
1019       BEGIN
1020          OPTIONSSELECTED   := [ SPACEAFTER,
1021                                 INDENTBYTAB,
1022                                 GOBBLESYMBOLS,
1023                                 CRAFTER ];
1024          DINDENTSYMBOLS    := [];
1025          GOBBLETERMINATORS := [ OFSYM ]
1026       END;
1027
1028    WITH PPOPTION [ CASEVARSYM ] DO
1029       BEGIN
1030          OPTIONSSELECTED   := [ SPACEAFTER,
1031                                 INDENTBYTAB,
1032                                 GOBBLESYMBOLS,
1033                                 CRAFTER ];
1034          DINDENTSYMBOLS    := [];
1035          GOBBLETERMINATORS := [ OFSYM ]
1036       END;
1037
1038    WITH PPOPTION [ OFSYM ] DO
1039       BEGIN
1040          OPTIONSSELECTED   := [ CRSUPPRESS,
1041                                 SPACEBEFORE ];
1042          DINDENTSYMBOLS    := [];
1043          GOBBLETERMINATORS := []
1044       END;
1045
1046    WITH PPOPTION [ FORSYM ] DO
1047       BEGIN
1048          OPTIONSSELECTED   := [ SPACEAFTER,
1049                                 INDENTBYTAB,
1050                                 GOBBLESYMBOLS,
1051                                 CRAFTER ];
1052          DINDENTSYMBOLS    := [];
1053          GOBBLETERMINATORS := [ DOSYM ]
1054       END;
1055
1056    WITH PPOPTION [ WHILESYM ] DO
1057       BEGIN
1058          OPTIONSSELECTED   := [ SPACEAFTER,
1059                                 INDENTBYTAB,
1060                                 GOBBLESYMBOLS,
1061                                 CRAFTER ];
1062          DINDENTSYMBOLS    := [];
1063          GOBBLETERMINATORS := [ DOSYM ]
1064       END;
1065
1066    WITH PPOPTION [ WITHSYM ] DO
1067       BEGIN
1068          OPTIONSSELECTED   := [ SPACEAFTER,
1069                                 INDENTBYTAB,
1070                                 GOBBLESYMBOLS,
1071                                 CRAFTER ];
1072          DINDENTSYMBOLS    := [];
1073          GOBBLETERMINATORS := [ DOSYM ]
1074       END;
1075
1076    WITH PPOPTION [ DOSYM ] DO
1077       BEGIN
1078          OPTIONSSELECTED   := [ CRSUPPRESS,
1079                                 SPACEBEFORE ];
1080          DINDENTSYMBOLS    := [];
1081          GOBBLETERMINATORS := []
1082       END;
1083
1084    WITH PPOPTION [ IFSYM ] DO
1085       BEGIN
1086          OPTIONSSELECTED   := [ SPACEAFTER,
1087                                 INDENTBYTAB,
1088                                 GOBBLESYMBOLS,
1089                                 CRAFTER ];
1090          DINDENTSYMBOLS    := [];
1091          GOBBLETERMINATORS := [ THENSYM ]
1092       END;
```

```
1093
1094    WITH PPOPTION [ THENSYM ] DO
1095       BEGIN
1096          OPTIONSSELECTED    := [ INDENTBYTAB,
1097                                  CRAFTER ];
1098          DINDENTSYMBOLS     := [];
1099          GOBBLETERMINATORS := []
1100       END;
1101
1102    WITH PPOPTION [ ELSESYM ] DO
1103       BEGIN
1104          OPTIONSSELECTED    := [ CRBEFORE,
1105                                  DINDENTONKEYS,
1106                                  DINDENT,
1107                                  INDENTBYTAB,
1108                                  CRAFTER ];
1109          DINDENTSYMBOLS     := [ IFSYM,
1110                                  ELSESYM ];
1111          GOBBLETERMINATORS := []
1112       END;
1113
1114    WITH PPOPTION [ ENDSYM ] DO
1115       BEGIN
1116          OPTIONSSELECTED    := [ CRBEFORE,
1117                                  DINDENTONKEYS,
1118                                  DINDENT,
1119                                  CRAFTER ];
1120          DINDENTSYMBOLS     := [ IFSYM,
1121                                  THENSYM,
1122                                  ELSESYM,
1123                                  FORSYM,
1124                                  WHILESYM,
1125                                  WITHSYM,
1126                                  CASEVARSYM,
1127                                  COLON,
1128                                  EQUALS ];
1129          GOBBLETERMINATORS := []
1130       END;
1131
1132    WITH PPOPTION [ UNTILSYM ] DO
1133       BEGIN
1134          OPTIONSSELECTED    := [ CRBEFORE,
1135                                  DINDENTONKEYS,
1136                                  DINDENT,
1137                                  SPACEAFTER,
1138                                  GOBBLESYMBOLS,
1139                                  CRAFTER ];
1140          DINDENTSYMBOLS     := [ IFSYM,
1141                                  THENSYM,
1142                                  ELSESYM,
1143                                  FORSYM,
1144                                  WHILESYM,
1145                                  WITHSYM,
1146                                  COLON,
1147                                  EQUALS ];
1148          GOBBLETERMINATORS := [ ENDSYM,
1149                                  UNTILSYM,
1150                                  ELSESYM,
1151                                  SEMICOLON ];
1152       END;
1153
1154    WITH PPOPTION [ BECOMES ] DO
1155       BEGIN
1156          OPTIONSSELECTED    := [ SPACEBEFORE,
1157                                  SPACEAFTER,
1158                                  GOBBLESYMBOLS ];
1159          DINDENTSYMBOLS     := [];
1160          GOBBLETERMINATORS := [ ENDSYM,
1161                                  UNTILSYM,
1162                                  ELSESYM,
1163                                  SEMICOLON ]
1164       END;
1165
1166    WITH PPOPTION [ OPENCOMMENT ] DO
1167       BEGIN
1168          OPTIONSSELECTED    := [ CRSUPPRESS ];
1169          DINDENTSYMBOLS     := [];
1170          GOBBLETERMINATORS := []
1171       END;
1172
1173    WITH PPOPTION [ CLOSECOMMENT ] DO
1174       BEGIN
1175          OPTIONSSELECTED    := [ CRSUPPRESS ];
1176          DINDENTSYMBOLS     := [];
1177          GOBBLETERMINATORS := []
1178       END;
```

```
1179
1180     WITH PPOPTION [ SEMICOLON ] DO
1181        BEGIN
1182           OPTIONSSELECTED    := [ CRSUPPRESS,
1183                                   DINDENTONKEYS,
1184                                   CRAFTER ];
1185           DINDENTSYMBOLS     := [ IFSYM,
1186                                   THENSYM,
1187                                   ELSESYM,
1188                                   FORSYM,
1189                                   WHILESYM,
1190                                   WITHSYM,
1191                                   COLON,
1192                                   EQUALS ];
1193           GOBBLETERMINATORS := []
1194        END;
1195
1196     WITH PPOPTION [ COLON ] DO
1197        BEGIN
1198           OPTIONSSELECTED    := [ SPACEAFTER,
1199                                   INDENTTOCLP ];
1200           DINDENTSYMBOLS     := [];
1201           GOBBLETERMINATORS := []
1202        END;
1203
1204     WITH PPOPTION [ EQUALS ] DO
1205        BEGIN
1206           OPTIONSSELECTED    := [ SPACEBEFORE,
1207                                   SPACEAFTER,
1208                                   INDENTTOCLP ];
1209           DINDENTSYMBOLS     := [];
1210           GOBBLETERMINATORS := []
1211        END;
1212
1213     WITH PPOPTION [ OPENPAREN ] DO
1214        BEGIN
1215           OPTIONSSELECTED    := [ GOBBLESYMBOLS ];
1216           DINDENTSYMBOLS     := [];
1217           GOBBLETERMINATORS := [ CLOSEPAREN ]
1218        END;
1219
1220     WITH PPOPTION [ CLOSEPAREN ] DO
1221        BEGIN
1222           OPTIONSSELECTED    := [];
1223           DINDENTSYMBOLS     := [];
1224           GOBBLETERMINATORS := []
1225        END;
1226
1227     WITH PPOPTION [ PERIOD ] DO
1228        BEGIN
1229           OPTIONSSELECTED    := [ CRSUPPRESS ];
1230           DINDENTSYMBOLS     := [];
1231           GOBBLETERMINATORS := []
1232        END;
1233
1234     WITH PPOPTION [ ENDOFFILE ] DO
1235        BEGIN
1236           OPTIONSSELECTED    := [];
1237           DINDENTSYMBOLS     := [];
1238           GOBBLETERMINATORS := []
1239        END;
1240
1241     WITH PPOPTION [ OTHERSYM ] DO
1242        BEGIN
1243           OPTIONSSELECTED    := [];
1244           DINDENTSYMBOLS     := [];
1245           GOBBLETERMINATORS := []
1246        END
1247
1248
1249  END; (* INITIALIZE *)
1250
1251
```

```
1252
1253 FUNCTION STACKEMPTY (* RETURNING *) : BOOLEAN;
1254
1255 BEGIN (* STACKEMPTY *)
1256
1257    IF TOP = 0
1258       THEN
1259          STACKEMPTY := TRUE
1260       ELSE
1261          STACKEMPTY := FALSE
1262
1263 END; (* STACKEMPTY *)
1264
1265
1266
1267 FUNCTION STACKFULL (* RETURNING *) : BOOLEAN;
1268
1269 BEGIN (* STACKFULL *)
1270
1271    IF TOP = MAXSTACKSIZE
1272       THEN
1273          STACKFULL := TRUE
1274       ELSE
1275          STACKFULL := FALSE
1276
1277 END; (* STACKFULL *)
1278
1279
1280
1281 PROCEDURE POPSTACK( (* RETURNING *) VAR INDENTSYMBOL : KEYSYMBOL;
1282                                       VAR PREVMARGIN   : INTEGER   );
1283
1284 BEGIN (* POPSTACK *)
1285
1286    IF NOT STACKEMPTY
1287       THEN
1288          BEGIN
1289
1290             INDENTSYMBOL := STACK[TOP].INDENTSYMBOL;
1291             PREVMARGIN   := STACK[TOP].PREVMARGIN;
1292
1293             TOP := TOP - 1
1294
1295          END
1296
1297       ELSE
1298          BEGIN
1299             INDENTSYMBOL := OTHERSYM;
1300             PREVMARGIN   := 0
1301          END
1302
1303 END; (* POPSTACK *)
1304
1305
1306
1307 PROCEDURE PUSHSTACK( (* USING *) INDENTSYMBOL : KEYSYMBOL;
1308                                    PREVMARGIN   : INTEGER   );
1309
1310 BEGIN (* PUSHSTACK *)
1311
1312    TOP := TOP + 1;
1313
1314    STACK[TOP].INDENTSYMBOL := INDENTSYMBOL;
1315    STACK[TOP].PREVMARGIN   := PREVMARGIN
1316
1317 END; (* PUSHSTACK *)
1318
1319
```

```
1320
1321 PROCEDURE WRITECRS( (* USING *)           NUMBEROFCRS : INTEGER;
1322                     (* UPDATING *)   VAR CURRLINEPOS : INTEGER;
1323                     (* WRITING TO *) VAR OUTPUTFILE  : TEXT    );
1324
1325 VAR
1326     I: INTEGER;
1327
1328
1329 BEGIN (* WRITECRS *)
1330
1331    IF NUMBEROFCRS > 0
1332       THEN
1333          BEGIN
1334
1335             FOR I := 1 TO NUMBEROFCRS DO
1336                WRITELN(OUTPUTFILE);
1337
1338             CURRLINEPOS := 0
1339
1340          END
1341
1342 END; (* WRITECRS *)
1343
1344
```

```
1345
1346 PROCEDURE INSERTCR( (* UPDATING *)   VAR CURRSYM    : SYMBOLINFO;
1347                     (* WRITING TO *) VAR OUTPUTFILE : TEXT       );
1348
1349 CONST
1350       ONCE = 1;
1351
1352
1353 BEGIN (* INSERTCR *)
1354
1355    IF CURRSYM^.CRSBEFORE = 0
1356       THEN
1357          BEGIN
1358
1359             WRITECRS( ONCE, (* UPDATING *)   CURRLINEPOS,
1360                             (* WRITING TO *) OUTPUTFILE  );
1361
1362             CURRSYM^.SPACESBEFORE := 0
1363
1364          END
1365
1366 END; (* INSERTCR *)
1367
1368
```

```
1369
1370 PROCEDURE INSERTBLANKLINE( (* UPDATING *)   VAR CURRSYM : SYMBOLINFO;
1371                            (* WRITING TO *) VAR OUTPUTFILE : TEXT  );
1372
1373 CONST
1374       ONCE  = 1;
1375       TWICE = 2;
1376
1377
1378 BEGIN (* INSERTBLANKLINE *)
1379
1380    IF CURRSYM^.CRSBEFORE = 0
1381       THEN
1382          BEGIN
1383
1384             IF CURRLINEPOS = 0
1385                THEN
1386                   WRITECRS( ONCE, (* UPDATING *)   CURRLINEPOS,
1387                                   (* WRITING TO *) OUTPUTFILE  )
1388                ELSE
1389                   WRITECRS( TWICE, (* UPDATING *)   CURRLINEPOS,
1390                                    (* WRITING TO *) OUTPUTFILE  );
1391
1392             CURRSYM^.SPACESBEFORE := 0
1393
1394          END
1395
1396       ELSE
1397          IF CURRSYM^.CRSBEFORE = 1
1398             THEN
1399                IF CURRLINEPOS > 0
1400                   THEN
1401                      WRITECRS( ONCE, (* UPDATING *)   CURRLINEPOS,
1402                                      (* WRITING TO *) OUTPUTFILE  )
1403
1404 END; (* INSERTBLANKLINE *)
1405
1406
```

```
1407
1408 PROCEDURE LSHIFTON( (* USING *) DINDENTSYMBOLS : KEYSYMSET );
1409
1410 VAR
1411     INDENTSYMBOL: KEYSYMBOL;
1412     PREVMARGIN  : INTEGER;
1413
1414
1415 BEGIN (* LSHIFTON *)
1416
1417    IF NOT STACKEMPTY
1418       THEN
1419          BEGIN
1420
1421             REPEAT
1422
1423                POPSTACK( (* RETURNING *) INDENTSYMBOL,
1424                                          PREVMARGIN    );
1425
1426                IF INDENTSYMBOL IN DINDENTSYMBOLS
1427                   THEN
1428                      CURRMARGIN := PREVMARGIN
1429
1430             UNTIL NOT(INDENTSYMBOL IN DINDENTSYMBOLS)
1431                    OR (STACKEMPTY);
1432
1433             IF NOT(INDENTSYMBOL IN DINDENTSYMBOLS)
1434                THEN
1435                   PUSHSTACK( (* USING *) INDENTSYMBOL,
1436                                          PREVMARGIN    )
1437
1438          END
1439
1440 END; (* LSHIFTON *)
1441
1442
1443
1444 PROCEDURE LSHIFT;
1445
1446 VAR
1447     INDENTSYMBOL: KEYSYMBOL;
1448     PREVMARGIN  : INTEGER;
1449
1450
1451 BEGIN (* LSHIFT *)
1452
1453    IF NOT STACKEMPTY
1454       THEN
1455          BEGIN
1456             POPSTACK( (* RETURNING *) INDENTSYMBOL,
1457                                       PREVMARGIN    );
1458             CURRMARGIN := PREVMARGIN
1459          END
1460
1461 END; (* LSHIFT *)
1462
1463
```

```
1464
1465 PROCEDURE INSERTSPACE( (* USING *)       VAR SYMBOL     : SYMBOLINFO;
1466                        (* WRITING TO *) VAR OUTPUTFILE : TEXT       );
1467
1468 BEGIN (* INSERTSPACE *)
1469
1470    IF CURRLINEPOS < MAXLINESIZE
1471       THEN
1472          BEGIN
1473
1474             WRITE(OUTPUTFILE, SPACE);
1475
1476             CURRLINEPOS := CURRLINEPOS + 1;
1477
1478             WITH SYMBOL^ DO
1479                IF (CRSBEFORE = 0) AND (SPACESBEFORE > 0)
1480                   THEN
1481                      SPACESBEFORE := SPACESBEFORE - 1
1482
1483          END
1484
1485 END; (* INSERTSPACE *)
1486
1487
```

```
1488
1489 PROCEDURE MOVELINEPOS( (* TO *)       NEWLINEPOS   : INTEGER;
1490                        (* FROM *) VAR CURRLINEPOS  : INTEGER;
1491                        (* IN *)   VAR OUTPUTFILE   : TEXT    );
1492
1493 VAR
1494    I: INTEGER;
1495
1496
1497 BEGIN (* MOVELINEPOS *)
1498
1499    FOR I := CURRLINEPOS+1 TO NEWLINEPOS DO
1500       WRITE(OUTPUTFILE, SPACE);
1501
1502    CURRLINEPOS := NEWLINEPOS
1503
1504 END; (* MOVELINEPOS *)
1505
1506
```

```
1507
1508 PROCEDURE PRINTSYMBOL( (* IN *)                  CURRSYM      : SYMBOLINFO;
1509                        (* UPDATING *)    VAR CURRLINEPOS : INTEGER;
1510                        (* WRITING TO *) VAR OUTPUTFILE  : TEXT       );
1511
1512 VAR
1513    I: INTEGER;
1514
1515
1516 BEGIN (* PRINTSYMBOL *)
1517
1518    WITH CURRSYM^ DO
1519       BEGIN
1520
1521          FOR I := 1 TO LENGTH DO
1522             WRITE(OUTPUTFILE, VALUE[I]);
1523
1524          CURRLINEPOS := CURRLINEPOS + LENGTH
1525
1526       END (* WITH *)
1527
1528 END; (* PRINTSYMBOL *)
1529
1530
```

```
1531
1532 PROCEDURE PPSYMBOL( (* IN *)              CURRSYM    : SYMBOLINFO;
1533                     (* WRITING TO *) VAR OUTPUTFILE : TEXT       );
1534
1535 CONST
1536       ONCE  = 1;
1537
1538 VAR
1539     NEWLINEPOS: INTEGER;
1540
1541
1542 BEGIN (* PPSYMBOL *)
1543
1544    WITH CURRSYM^ DO
1545       BEGIN
1546
1547          WRITECRS( (* USING *)       CRSBEFORE,
1548                    (* UPDATING *)    CURRLINEPOS,
1549                    (* WRITING TO *) OUTPUTFILE  );
1550
1551          IF (CURRLINEPOS + SPACESBEFORE > CURRMARGIN)
1552             OR (NAME IN [ OPENCOMMENT, CLOSECOMMENT ])
1553             THEN
1554                NEWLINEPOS := CURRLINEPOS + SPACESBEFORE
1555             ELSE
1556                NEWLINEPOS := CURRMARGIN;
1557
1558          IF NEWLINEPOS + LENGTH > MAXLINESIZE
1559             THEN
1560                BEGIN
1561
1562                   WRITECRS( ONCE, (* UPDATING *)    CURRLINEPOS,
1563                                   (* WRITING TO *) OUTPUTFILE  );
1564
1565                   IF CURRMARGIN + LENGTH <= MAXLINESIZE
1566                      THEN
1567                         NEWLINEPOS := CURRMARGIN
1568                      ELSE
1569                         IF LENGTH < MAXLINESIZE
1570                            THEN
1571                               NEWLINEPOS := MAXLINESIZE - LENGTH
1572                            ELSE
1573                               NEWLINEPOS := 0
1574
1575                END;
1576
1577          MOVELINEPOS( (* TO *)     NEWLINEPOS,
1578                       (* FROM *)   CURRLINEPOS,
1579                       (* IN *)     OUTPUTFILE  );
1580
1581          PRINTSYMBOL( (* IN *)          CURRSYM,
1582                       (* UPDATING *)    CURRLINEPOS,
1583                       (* WRITING TO *) OUTPUTFILE  )
1584
1585       END (* WITH *)
1586
1587 END; (* PPSYMBOL *)
1588
1589
```

```
1590
1591 PROCEDURE RSHIFTTOCLP( (* USING *) CURRSYM : KEYSYMBOL );
1592    FORWARD;
1593
1594 PROCEDURE GOBBLE( (* SYMBOLS FROM *) VAR INPUTFILE    : TEXT;
1595                   (* UP TO *)           TERMINATORS : KEYSYMSET;
1596                   (* UPDATING *)     VAR CURRSYM,
1597                                         NEXTSYM      : SYMBOLINFO;
1598                   (* WRITING TO *)   VAR OUTPUTFILE  : TEXT       );
1599
1600 BEGIN (* GOBBLE *)
1601
1602    RSHIFTTOCLP( (* USING *) CURRSYM^.NAME );
1603
1604    WHILE NOT(NEXTSYM^.NAME IN (TERMINATORS + [ENDOFFILE])) DO
1605       BEGIN
1606
1607          GETSYMBOL( (* FROM *)       INPUTFILE,
1608                     (* UPDATING *)   NEXTSYM,
1609                     (* RETURNING *) CURRSYM    );
1610
1611          PPSYMBOL( (* IN *)          CURRSYM,
1612                    (* WRITING TO *) OUTPUTFILE )
1613
1614       END; (* WHILE *)
1615
1616    LSHIFT
1617
1618 END; (* GOBBLE *)
1619
1620
```

```
1621
1622 PROCEDURE RSHIFT( (* USING *) CURRSYM : KEYSYMBOL );
1623
1624 BEGIN (* RSHIFT *)
1625
1626    IF NOT STACKFULL
1627       THEN
1628          PUSHSTACK( (* USING *) CURRSYM,
1629                                   CURRMARGIN);
1630
1631    IF CURRMARGIN < SLOWFAIL1
1632       THEN
1633          CURRMARGIN := CURRMARGIN + INDENT1
1634       ELSE
1635          IF CURRMARGIN < SLOWFAIL2
1636             THEN
1637                CURRMARGIN := CURRMARGIN + INDENT2
1638
1639 END; (* RSHIFT *)
1640
1642
1643 PROCEDURE RSHIFTTOCLP;
1644
1645 BEGIN (* RSHIFTTOCLP *)
1646
1647    IF NOT STACKFULL
1648       THEN
1649          PUSHSTACK( (* USING *) CURRSYM,
1650                                   CURRMARGIN);
1651
1652    CURRMARGIN := CURRLINEPOS
1653
1654 END; (* RSHIFTTOCLP *)
1655
1656
```

```
1657
1658 BEGIN (* PRETTYPRINT *)
1659
1660     INITIALIZE( INPUTFILE,  OUTPUTFILE, TOP,          CURRLINEPOS,
1661                 CURRMARGIN, KEYWORD,    DBLCHARS,     DBLCHAR,
1662                 SGLCHAR,    RECORDSEEN, CURRCHAR,     NEXTCHAR,
1663                 CURRSYM,    NEXTSYM,    PPOPTION );
1664
1665     CRPENDING := FALSE;
1666
1667     WHILE (NEXTSYM^.NAME <> ENDOFFILE) DO
1668        BEGIN
1669
1670           GETSYMBOL( (* FROM *)       INPUTFILE,
1671                      (* UPDATING *)   NEXTSYM,
1672                      (* RETURNING *) CURRSYM    );
1673
1674           WITH PPOPTION [CURRSYM^.NAME] DO
1675              BEGIN
1676
1677                 IF (CRPENDING AND NOT(CRSUPPRESS IN OPTIONSSELECTED))
1678                   OR (CRBEFORE IN OPTIONSSELECTED)
1679                    THEN
1680                       BEGIN
1681                          INSERTCR( (* USING *)       CURRSYM,
1682                                    (* WRITING TO *) OUTPUTFILE );
1683                          CRPENDING := FALSE
1684                       END;
1685
1686                 IF BLANKLINEBEFORE IN OPTIONSSELECTED
1687                    THEN
1688                       BEGIN
1689                          INSERTBLANKLINE( (* USING *)       CURRSYM,
1690                                           (* WRITING TO *) OUTPUTFILE );
1691                          CRPENDING := FALSE
1692                       END;
1693
1694                 IF DINDENTONKEYS IN OPTIONSSELECTED
1695                    THEN
1696                       LSHIFTON(DINDENTSYMBOLS);
1697
1698                 IF DINDENT IN OPTIONSSELECTED
1699                    THEN
1700                       LSHIFT;
1701
1702                 IF SPACEBEFORE IN OPTIONSSELECTED
1703                    THEN
1704                       INSERTSPACE( (* USING *)       CURRSYM,
1705                                    (* WRITING TO *) OUTPUTFILE );
1706
1707                 PPSYMBOL( (* IN *)          CURRSYM,
1708                           (* WRITING TO *) OUTPUTFILE );
1709
1710                 IF SPACEAFTER IN OPTIONSSELECTED
1711                    THEN
1712                       INSERTSPACE( (* USING *)       NEXTSYM,
1713                                    (* WRITING TO *) OUTPUTFILE );
1714
1715                 IF INDENTBYTAB IN OPTIONSSELECTED
1716                    THEN
1717                       RSHIFT( (* USING *) CURRSYM^.NAME );
1718
1719                 IF INDENTTOCLP IN OPTIONSSELECTED
1720                    THEN
1721                       RSHIFTTOCLP( (* USING *) CURRSYM^.NAME );
1722
1723                 IF GOBBLESYMBOLS IN OPTIONSSELECTED
1724                    THEN
1725                       GOBBLE( (* SYMBOLS FROM *) INPUTFILE,
1726                               (* UP TO *)        GOBBLETERMINATORS,
1727                               (* UPDATING *)     CURRSYM,
1728                                                  NEXTSYM,
1729                               (* WRITING TO *)   OUTPUTFILE        );
1730
1731                 IF CRAFTER IN OPTIONSSELECTED
1732                    THEN
1733                       CRPENDING := TRUE
1734
1735              END (* WITH *)
1736
1737        END; (* WHILE *)
1738
1739     IF CRPENDING
1740        THEN
1741           WRITELN(OUTPUTFILE)
1742
1743 END.
```