Table of Contents

INTRODUCTION

PLIFORM is a PL/I program which reformats PL/I source code. It was developed at Cornell University in response to their needs and the inadequacies of the then available formatting tools. Several problems were identified with earlier PL/I formatters (PL/I Checker FORMAT option and NEATER2):

- Handling of comments ranges from bad to horrible.
- They do not provide for differing record formats on input and output (i.e. Fixed -> varying).
- They do not handle PL/I macro statements in a reasonable way.
- The Checker formatter is not designed to produce a new source file as output. It assumes that the output is to be printed. The Checker formatter converts all of the input source to upper case.
- Last, but far from least, they format according to their author's concept of what PL/I code should look like. There is no "correct" format for PL/I code; therefore to be correct, a formatter must be adaptable to the standards preferred by each programmer who will use it.

In order to have a formatter without these limitations, it was decided to build a table-driven formatter capable of being tailored by the end user. The result of this effort is PLIFORM, a truly flexible formatter for PL/I.

This is an initial OS/2 port of PLIFORM. The version is “1.2” because it includes minor changes from the original mainframe version 1.1. For information on the port see “NOTES ON THE OS/2 PORT”.

USING PLIFORM

To run PLIFORM, type the command:

```
PLIFORM [ parm-file ]  input-file  output-file
```

- 'parm' file is optional. If present it specifies the name of the parameter file. If omitted the name defaults to 'PLIFORM.PRM'.
- 'input-file' is the name of the PL/I source program to be reformatted.
- 'output-file' is the name of the reformatted source program.

PLIFORM has several options which can be specified at run time. The options are specified in a user-created parameter file. See "SEARCH ORDER" for a description of where this file may be located. The parameter file is free-form. The only restriction is that words must not be split between lines. If the parameter file is not found, PLIFORM will use built-in defaults.

DEBUG Causes one of several debug listings to be produced on the output file SYSPRINT.

DEBUG(1) will cause a listing to be produced which shows the step-by-step use of the formatting rules.

DEBUG(2) will cause a listing showing how line splits were arrived at.

DEBUG(4) and
DEBUG(5) are intended for use by personnel responsible for the maintenance of PLIFORM and produce (large) traces of internal logic flows. If you have a problem with the formatter, please save a listing of the input source, DEBUG 1, 4, and 5 listings, and the formatted output.

The default is that no debug output will be generated.

CASE Determines how upper and lower case will be handled.

CASE(M) does nothing to the code [default].

CASE(U) will cause all code (except text in quoted strings and comments) to be put into upper case.

CASE(C) causes all tokens to have their first character in upper case and the rest in lower case.

CASE(L) causes all tokens to be put into lower case.

SEQ or **NOSEQ** Specifies the location of sequence numbering in the output. The format is SEQ fcol lcol incr gang), where fcol is the first column of the sequence field, lcol is the last column of the sequence field, incr is the increment to be used, and gang is a character string to be gang punched over the beginning of the field. The default is SEQ(73 80 10).

NOSEQ specifies that sequence numbers will not be generated.

MAR Specifies the margins of the input. This option is identical to the MAR option on the PL/I compiler: MAR(2,72).

OMAR Specifies the margins of the output. The format is the same as for MAR.

LRECL Specifies the Maximum record length of the output file, exclusive of the trailing CR/LF..

RECFM Specifies the format of the output file. RECFM(V) specifies standard variable-length lines. RECFM(F) requests lines padded with blanks to LRECL.

DECLARE or **NODCL** Specify whether DECLARE statements should be formatted

DECLARE (or DCL) is the default, and indicates that DECLARE statements should be formatted.

NODCL will suppress formatting of DECLARE statements.

PRINT and **FILE** are mutually exclusive, and control how PL/I listing control statements %CONTROL(FORMAT); %CONTROL(NOFORMAT); and %PAGE;) are handled. If the FILE option is specified, all of the listing control statements are preserved in the file, and only %CONTROL(NOFORMAT); and %CONTROL(FORMAT); have any effect. If the listing control statements will appear in the output file; they will all be used to control the listing format. Note that the listing control statements are subject to the restrictions listed in the PL/I language reference; they must be on a separate line, must not be in a comment, and must not be embedded in a statement. FILE is the default.

CREATING FORMATTING RULES

The formatting performed by PLIFORM is governed by three dynamically loaded tables, FORMCTL.TABLE, SPACTL.TABLE, and SPLTCTL.TABLE. These tables control overall formatting, token-to-token spacing, and line splitting. Utility routines (FORMSET, SPACSET, and SPLTSET) are provided to transform these tables into a form which can be easily used by PLIFORM. The instructions for running these utilities can be found in one of the appendices.

SEARCH ORDER

The files FORMCTL.TABLE, SPLTCTL.TABLE, and SPACTL.TABLE, as well as the parameter file PLIFORM.PRM (only if the user does not name this file on the command line) will be searched for in order in the following locations:

1. The current working directory.
2. The directory where the PLIFORM program is located.
3. The directories named in the DPATH environment variable.

FORMCTL

FORMCTL is the table which controls the formatting of the PL/I source. It consists of user supplied actions to be taken in each of 110 situations which can arise during the formatting of PL/I code. The actual structure of the table is shown below.

```
/*******************************************/   /*****************************/
/*                                         */   /*                           */
/*   Current situation                     */   /*   Action(s) to be taken   */
/*                                         */   /*                           */
                                          .
                                          .
                                          .
/*   # END                                 25   */        SKIP,COLREL,-3
/*   # PROCEDURE ... ;                     26   */        PUSH,SETIN,7,COLREL,-3
/*   # SELECT ... ;                        27   */        SKIP,INCR,+3
/*   THEN # comment                        28   */        INCOLPLUS
/*   THEN comment #                        29   */        SKIP
/*   THEN # (cond prefix) :                30   */        COL,1
/*   THEN # label :                        31   */        COL,1
/*   THEN # IF                             32   */        SKIP
/*   THEN # BEGIN                          33   */        SKIP,INCR,+3
                                          .
                                          .
                                          .
```

The pound sign (#) indicates the current position in the formatting process. For example, in the last line of the segment of the table shown above we have indicated that when the keyword THEN is immediately followed by the keyword BEGIN, the formatter should skip to a new line and increment the indentation level by three columns. The commands modify two variables in the formatter, the indentation level (which I will refer to as INDENT) and the current output column position I (which I will refer to as COL).

SKIP Causes a skip to a new line (if we are not currently on a blank line). COL is set to the current value of INDENT.

INCR causes INDENT to be incremented by the argument value. The argument may be positive or negative.

COL causes COL to be set to the argument value. If the argument value is less than COL and we are not on a blank line, a skip will occur.

COLPUS causes COL to be set to the argument value if it is greater than the current value of COL.

PUSH causes the current value of INDENT to be stacked.

POP pops the most recently stored value of INDENT.

SETIN sets the value of INDENT to the argument value.

COLREL sets the value of COL to the current value of INDENT plus the argument value. The argument may be positive or negative. Can cause a skip if the command will cause the value of COL to decrease.

INCOL causes COL to be set to the input column location of the following token. This allows the formatter to put comments back where it found them, if desired.

INCOLPLUS causes COL to be set to the input column location only if this does not decrease the value of COL.

STORE stores the current value of INDENT in a special variable called DCL Inset.

STOREL stores the current value of INDENT plus the argument in DCL INSET.

SLEVEL sets INDENT to the DCL INSET plus the product of the argument and the value of the next token. This is used to format structures and may ONLY be used where the next token is a number. The argument corresponds to the number of columns you wish to indent each logical level in the structure.

RLEVEL resets INDENT to DCL INSET

Several of the entries in the table refer to logical positions on the output rather than physical positions. These entries are distinguished by not having a pound sign. For example, the first entry in the table is:

```
/*      endthen            01   */       INCR,-3,COLREL,-3
```

This series of commands will be executed each time the formatter detects the end of a THEN clause. Three entries in the table (split, splitcc and splitattr) are executed when the formatter finds that line split is necessary. The "split" entry is used for most line splits. The "splitcc" entry is executed when a line split is caused by a carriage control character in the input file. The "Splitattr" entry is executed when an attribute list is split in a DECLARE statement. Entries 82 through 109 pertain only to DECLARE statements.

To assist the user in debugging new FORMCTL tables, a trace of the formatting process can be obtained. This is done by specifying the parameter DEBUG(1). The listing produced shows the values of INDENT and COL before execution of each formatting rule, the formatting rule itself, the values of INDENT and COL after application of the rule, and then the tokens written to the output file before then next formatting rule was invoked. This detailed trace is extremely valuable for sorting out the sometimes very subtle interactions between the various formatting rules. A complete set of formatting rules are given in the appendices. It is recommended that you study them carefully and try to understand why they work before attempting to build a new set of rules.

SPACTL

SPACTL is the table which controls token-to-token spacing. It allows the user to specify, for each possible combination of types of tokens, whether there should never be a blank between the tokens, always be a blank between the tokens, or whether it should be left as it was input. In order to facilitate specification of this table, the legal PL/I tokens have been divided into 34 groups, with a mnemonic name assigned to each group.

```
Mnemonic        Tokens

COMMENT         PL/I comments
WORD            PL/I identifiers and keywords other
                than labels and those specifically listed.
NUMBER          all numbers
CHARSTR         quoted character strings
BITSTR          all bit strings
LOGOP           logical operators  (& | ¬)
NUMOP           numeric operators (+ - * / **)
RELOP           relational operators  (<  <= = ¬= >= >)
=               assignment (=)
->              ->
.               .
;               ;
:               :
(               (
)               )
,               ,
%               %
IF              PL/I keyword
THEN            PL/I keyword
ELSE            PL/I keyword
WHEN            PL/I keyword
ON              PL/I keyword
DCL             PL/I keyword (DCL or DECLARE)
OTHER           PL/I keyword (OTHER or OTHERWISE)
DO              PL/I keyword
END             PL/I keyword
ENDL            PL/I keyword END (when followed by a label, i.e. END PHRASE;)
SELECT          PL/I keyword
LABEL           PL/I label
BEGIN           PL/I keyword
PROC            PL/I keyword (PROC or PROCEDURE)
```

In addition, an asterisk may be used to indicate a token of unspecified type. The structure of the table is the mnemonic for the preceding token, the mnemonic for the succeeding token, and the rule (NOBLANK, BLANK or SAME) to be used for determining their spacing. For example the rules

```
WORD            ->        NOBLANK
->              WORD      NOBLANK
```

would indicate that pointer references should never contain blanks while the rules

```
*          +          BLANK
=          *          BLANK
```

would indicate that all assignment operators should be surrounded by blanks. This table is searched from top to bottom for a match. In order to make sure that all combinations are covered, the last entry in the table should be

```
*          *          SAME
```

SPLTCTL

SPLTCTL is the table which controls selection of the split-point for statement which do not fit on a single line. The input for this table looks very much like the input for SPACTL, except that a weight (range 0-15, one decimal place allowed) is supplied instead of the BLANK, NOBLANK or SAME. The weight describes the "badness" of that position as a break point. For example, the line

```
(          *          15.0
```

tells PLIFORM that just after a parenthesis is a terrible place to break a line. Two special entries should be supplied for this table. They look like this:

```
SPACEWT         0.2
PARENWT         3.6
```

The SPACEWT value provided a weight which is applied for each column scanned in order to gently force splits toward the right hand end of the line. PARENWT value is used to decrease the probability of breaking a line inside a pair of parenthesis. The interactions of these various weights is subtle and will be explained more clearly in the next version of this manual. For now I would suggest printing the sample SPLTCTL INPUT file and the DEBUG(2) listings for a program which involves lots of line splits and study it.

NOTES ON THE OS/2 PORT

This is the initial port of PLIFORM to OS2. The program was modified as little as possible from the mainframe original. The changes consisted of adding OPEN statements for all files used in order to specify the appropriate OS/2 options, adding code to use a parameter file for the options rather than the command-line ('parm' in mainframe terms), adding the CASE(L) option, and accepting ASCII input rather than EBCDIC.

LANGUAGE LEVEL

The code should be enhanced to recognize Enterprise PL/I keywords such as PACKAGE and DEFINE. This would require expanding several tables in the program. In order to do this *right*, FORMCTL, SPACTL, and SPLTCTL should be changed to output a header record containing the table dimensionality. The tables in PLIFORM should be changed to CONTROLLED using this information, and several references using constant upper bounds should be changed to HBOUND(...).

SPECIAL CHARACTERS

This version of PLIFORM recognizes the following code points for special characters, corresponding to code page 819:

OR	**U+007C (dec 124)**
NOT	**U+00AC (dec 172)**

OS/2 PL/I allows the programmer to specify up to seven symbols each to be used to represent OR and NOT, for example, "^" and/or "!" for NOT. PLIFORM needs to allow for the same variations through the parameter file, by, for example, adding a parameter NOT, such as "`NOT(^!\)`", or better yet "`NOT(^!\:¬)`", where the ":¬" instructs the program to translate "^", "!", and "\" to "¬" on output.

Enjoy, and GOOD LUCK!
Peter Flass <Peter_Flass@Yahoo.com>
2 August, 2006

APPENDIX A. RUNNING THE FORMSET, SPACSET, and SPLTSET UTILITIES

FORMSET

The supplied input to FORMSET is the file FORMCTL.INPUT. To run FORMSET, either update this file or create a new one and type the command:

```
FORMSET input-file output-file
```

The output will be placed in 'output-file' which should be named FORMCTL.TABLE and placed in an appropriate directory for use by PLIFORM. See “SEARCH ORDER”.

SPACSET

The supplied input to SPACSET is the file SPACTL.INPUT. To run SPACSET, either update this file or create a new one and type the command:

```
SPACSET input-file output-file
```

The output will be placed in 'output-file' which should be named SPACTL.TABLE and placed in an appropriate directory for use by PLIFORM. See “SEARCH ORDER”.

SPLTSET

The supplied input to SPLTSET is the file SPLTCTL.INPUT. To run SPLTSET, either update this file or create a new one and type the command:

```
SPLTSET input-file output-file
```

The output will be placed in 'output-file' which should be named SPLTCTL.TABLE and placed in an appropriate directory for use by PLIFORM. See “SEARCH ORDER”.

APPENDIX B. EXAMPLE FORMCTL TABLE

```
column:
0---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
/************************************/  /**************************/
/*                                  */  /*                          */
/*  Current situation               */  /*  Action(s) to be taken   */
/*                                  */  /*                          */
/*  endthen                      01 */      INCR,-3,COLREL,-3
/*  endelse                      02 */      INCR,-3,COLREL,-3
/*  endwhen                      03 */      INCR,-3,COLREL,-3
/*  endother                     04 */      INCR,-3,COLREL,-3
/*  endon                        05 */      INCR,-3,COLREL,-3
/*  enddo                        06 */      INCR,-3,COLREL,-3
/*  endbegin                     07 */      INCR,-3,COLREL,-3
/*  endselect                    08 */      INCR,-3,COLREL,-3
/*  endproc                      09 */      POP,COLREL,0
/*  #  comment                   10 */      INCOL
/*  comment  #                   11 */      COLREL,0
/*  ;  #                         12 */      0
/*  # (cond prefix):             13 */      COL,1
/*  (cond prefix): #             14 */      COLREL,0
/*  # label:                     15 */      COL,1
/*  label: #                     16 */      COLREL,0
/*  # IF                        17 */      SKIP
/*  IF cond # THEN               18 */      INCR,+3
/*  # ELSE                      19 */      SKIP,INCR,+3
/*  # WHEN                      20 */      SKIP,INCR,+3
/*  # BEGIN                     21 */      SKIP,INCR,+3
/*  # DECLARE                   22 */      SKIP,PUSH,INCR,+3,STORREL,0
/*  # OTHERWISE                 23 */      SKIP,INCR,+3
/*  # DO ... ;                  24 */      SKIP,INCR,+3
/*  # END                       25 */      SKIP,COLREL,-3
/*  # PROCEDURE ... ;           26 */      PUSH,SETIN,7,COLREL,-3
/*  # SELECT ... ;              27 */      SKIP,INCR,+3
/*  THEN # comment               28 */      INCOLPLUS
/*  THEN comment #               29 */      SKIP
/*  THEN # (cond prefix):        30 */      COL,1
/*  THEN # label:                31 */      COL,1
/*  THEN # IF                    32 */      SKIP
/*  THEN # BEGIN                 33 */      SKIP,INCR,+3
/*  THEN # DO                    34 */      COLREL,0,INCR,+3
/*  THEN # SELECT                35 */      SKIP,INCR,+3
/*  THEN # ;                     36 */      0
/*  THEN # simple statement      37 */      SKIP
/*  ELSE # comment               38 */      INCOLPLUS
/*  ELSE comment #               39 */      SKIP
/*  ELSE # (cond prefix):        40 */      COL,1
/*  ELSE # label:                41 */      COL,1
/*  ELSE # IF                    42 */      SKIP
/*  ELSE # BEGIN                 43 */      SKIP,INCR,+3
/*  ELSE # DO                    44 */      SKIP,INCR,+3
```

```
/*  ELSE # SELECT                  45 */      SKIP,INCR,+3
/*  ELSE # ;                       46 */      0
/*  ELSE # simple statement        47 */      SKIP
/*  OTHERWISE # comment            48 */      INCOLPLUS
/*  OTHERWISE comment #            49 */      SKIP
/*  OTHERWISE # (cond prefix):     50 */      COL,1
/*  OTHERWISE # label:             51 */      COL,1
/*  OTHERWISE # IF                 52 */      SKIP
/*  OTHERWISE # BEGIN              53 */      SKIP,INCR,+3
/*  OTHERWISE # DO                 54 */      SKIP,INCR,+3
/*  OTHERWISE # SELECT             55 */      SKIP,INCR,+3
/*  OTHERWISE # ;                  56 */      0
/*  OTHERWISE # simple statement   57 */      SKIP
/*  WHEN # comment                 58 */      INCOLPLUS
/*  WHEN comment #                 59 */      SKIP
/*  WHEN # (cond prefix):          60 */      COL,1
/*  WHEN # label:                  61 */      COL,1
/*  WHEN # IF                      62 */      SKIP
/*  WHEN # BEGIN                   63 */      SKIP,INCR,+3
/*  WHEN # DO                      64 */      SKIP,INCR,+3
/*  WHEN # SELECT                  65 */      SKIP,INCR,+3
/*  WHEN # ;                       66 */      0
/*  WHEN # simple statement        67 */      SKIP
/*  ON ... # comment               68 */      INCOLPLUS
/*  ON ... comment #               69 */      SKIP
/*  ON ... # BEGIN                 70 */      SKIP,INCR,3
/*  ON ... # ;                     71 */      0
/*  ON ... # simple statement      72 */      0
/*  enddeclare                     73 */      POP,COLREL,0
/*  split                          74 */      SKIP,COLREL,+3
/*  splitcc                        75 */      SKIP
/*  # ON                           76 */      SKIP,INCR,+3
/*  THEN ... # ON                  77 */      SKIP
/*  ELSE ... # ON                  78 */      SKIP
/*  OTHERWISE ... # ON             79 */      SKIP
/*  WHEN ... # ON                  80 */      SKIP
/*  # simple statement             81 */      SKIP
/*  DECLARE # comment              82 */      SKIP
/*  DECLARE # id                   83 */      SKIP
/*  DECLARE # (                    84 */      SKIP
/*  DECLARE # NUM                  85 */      SKIP,SLEVEL,2,COLREL,-2
/*  , # comment                    86 */      INCOL
/*  , # id                         87 */      RLEVEL,COLREL,0
/*  , # (                          88 */      SKIP
/*  , # num                        89 */      SLEVEL,2,COLREL,-2,0
/*  ( # comment                    90 */      INCOL
/*  ( # id                         91 */      0
/*  ( # num                        92 */      SLEVEL,2,COLREL,-2,0
/*  comment # comment              93 */      0
/*  comment # id                   94 */      COLREL,0
/*  comment # (                    95 */      0
/*  comment # num                  96 */      SLEVEL,2,COLREL,-2,0
/*  comment # attrlist             97 */      COLPLUS,30
```

```
/*  comment # )                    98 */      0
/*  comment # ,                    99 */      0
/*  comment # ;                   100 */      0
/*  num # comment                 101 */      INCOL
/*  num # id                      102 */      COLREL,0
/*  num # (                       103 */      COLREL,0
/*  id # comment                  104 */      INCOL
/*  id # attrlist                 105 */      COLPLUS,30
/*  attrlist # comment            106 */      INCOL
/*  ) # comment                   107 */      INCOL
/*  ) # attrlist                  108 */      COLPLUS,30
/*  SPLITATTR                     109 */      COL,30
/*  %...; # COMMENT               110 */      INCOL
/*                                  */
/************************************/
```

APPENDIX C. EXAMPLE SPACTL TABLE

```
column:
0---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
  %       WORD          NOBLANK
  )         .           NOBLANK
 WORD       .           NOBLANK
  .         WORD        NOBLANK
  (         *           NOBLANK
  *         )           NOBLANK
  *         =           BLANK
  =         *           BLANK
  *       RELOP         BLANK
RELOP       *           BLANK
  *       LOGOP         BLANK
LOGOP       *           BLANK
  *       NUMOP         BLANK
NUMOP       *           BLANK
  *       CHAROP        BLANK
CHAROP      *           BLANK
  *         ;           NOBLANK
  *         ,           NOBLANK
  ,         *           BLANK
  *         :           NOBLANK
  :         *           NOBLANK
  *         *           SAME
```

APPENDIX D. EXAMPLE SPLTCTL TABLE

```
column:
0---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
SPACEWT               0.2
PARENWT               3.6
  (         *          +13
  *         )          +13
  *         =           +5
  =         *           0
  *       RELOP         +3
RELOP       *           0
  *       LOGOP         +3
LOGOP       *           0
  *       NUMOP         +3
NUMOP       *           0
  *       CHAROP        +3
CHAROP      *           0
  *         ;           +8
  *         ,           +5
  ,         *           0
  *         :          +11
  :         *           +5
  *         .          +11
  .         *           +5
  )         (           0
  *         (           +4
  )       NUMBER        +3
  )         *           0
  ->        *           +7
  *         ->          +7
  %       WORD          +11
WORD    CHARSTR         +12
  *         *           0
```