USER'S GUIDE TO RELEASE 2, PL/CS

T. Bishop and R. Conway

TR 77-315

Part I

Summary of the PL/CS Subset of PL/I

The following is a summary of only that portion of the PL/I language that is included in PL/CS. For a description of the full PL/C and PL/I languages see the PL/C *User's Guide* and the PL/I(F) Language Reference Manual (Order No. C28-8201).

PL/CS is not quite a strict subset of PL/I or PL/C. Neither the ASSERT statement, nor the READONLY attribute of PL/CS is included in PL/I or the current Release of PL/C. (PL/C also does not have the LEAVE, SELECT or DO UNTIL statements.) When either of these features is used, a PL/CS program is incompatible with PL/I and PL/C. If compatibility is important, ASSERT and READONLY can be sheltered in pseudo-comments, and LEAVE, SELECT and DO UNTIL can be converted to standard PL/C constructs.

This summary simply indicates what elements of the language are included in the subset. It shows the general form of each construction included, but does not attempt to teach what it does or how it should be used. It assumes the reader knows some subset of PL/I or is learning one from another source.

In the description below, words in *capital letters* are keywords, and must be given exactly as shown. Words in *lower-case* letters represent varying information, to be specified by the programmer.

Program

```
*PL/CS
 /* Comment describing function of program */
 procedure-name: PROCEDURE OPTIONS(MAIN);
     Declaration of variables:
     Statements:
     END procedure-name;
*PROCESS options
 /* Comment describing action of procedure */
 procedure-name: PROCEDURE (parameters);
     Declarations of parameters:
     Declarations of local variables;
     Statements;
     END procedure-name;
*PROCESS options
 /* Comment describing action of function */
 function-name: PROCEDURE (parameters) RETURNS (attributes);
      Declarations of parameters READONLY;
      Declarations of local variables;
      Statements;
      END function-name;

 ...
*DATA
     Data list
```

In addition to the program structure shown above, most installations require one or more control cards before the *PL/CS card, and one or more control cards after the data list. These cards vary from one installation to another -- get local instructions.

Declarations

```
DECLARE (names) attributes:
DECLARE (name(lb1:ub1,lb2:ub2,... )) attributes;
```

Attributes

```
FIXED
FLOAT
CHARACTER (length) VARYING
BIT (1)
STATIC
EXTERNAL
INITIAL(constant)
INITIAL(c1, c2, ... )
INITIAL(c1,(r1)c2,(r2)(c3,c4,... ),... )
READONLY
```

Statements

```
numeric variable = arithmetic expression;

string variable = string expression;

SUBSTR(s, f, n) = string expression;

array = array;

array = constant;

bit variable = (condition);

IF (condition)
     THEN statement;

IF (condition)
     THEN statement1;
     ELSE statement2;

DO;
     body;
     END;

select-name: SELECT;
     WHEN (condition1) statement1;
     WHEN (condition2) statement2;
     ...
     OTHERWISE statementn;
     END select-name;

loop-name: DO WHILE (condition);
     body;
     END loop-name;

loop-name: DO UNTIL (condition);
     body;
     END loop-name;

loop-name: DO index var = expr1 TO expr2 BY expr3;
     body;
     END loop-name;

LEAVE loop-name;

GOTO target-label;
...
target-label:;

ON ENDFILE GOTO DATAEND;
...
DATAEND:;

CALL procedure-name;
```

```
CALL procedure-name(arguments);

RETURN;

RETURN(expression);

GET LIST (variables);
GET DATA (variables);
GET DATA;
GET EDIT (variables) (format items);

PUT SKIP (expr) LIST (expressions);
PUT SKIP (expr) DATA (variables);
PUT SKIP (expr) EDIT (values) (format items);

ASSERT (condition);

ASSERT  (cond) FOR ALL index var = expr1 TO expr2 BY expr3;

ASSERT (cond) FOR SOME index var = expr1 TO expr2 BY expr3;
```

Operators

```
+     -     *     /     **     ||
```

Relations

```
=     ¬=     >     ¬>     >=     <     ¬<     <=
```

Boolean Operators

```
&     |     ¬
```

Constants

integers (fixed-point)
decimal and exponential (floating-point)
character strings
'1'B, '0'B ("true" and "false" in conditions)

<u>Built-in Functions</u> (see Section I.C)

ABS(x)
ALL(bit array)
ANY(bit array)
ATAN(x) and ATAN(x1,x2)
ATAND(x1,x2)
ATANH(x)
CEIL(x)
COS(x)
COSD(x)
COSH(x)
DATE
DIM(array,i)
ERF(x)
ERFC(x)
EXP(x)
FIXED(x)
FLOAT(x)
FLOOR(x)
HBOUND(array,i)
HIGH(i)
INDEX(s1,s2)
LBOUND(array,i)
LENGTH(s)
LOG(x)
LOG10(x)
LOG2(x)
LOW(i)
MAX(x1,x2,...,xn)
MIN(x1,x2,...,xn)
MOD(x,y)
PROD(array)
RAND(x) (not a PL/I function)
REPEAT(s,i)
ROUND(x,n)
SIGN(x)
SIN(x)
SIND(x)
SINH(x)
SQRT(x)
STRING(s)
SUBSTR(s,f,n) and SUBSTR(s,f)
SUM(array)
TAN(x)
TAND(x)
TANH(x)
TIME
TRANSLATE(s1,s2,s3)
TRUNC(x)
VERIFY(s1,s2)

Pseudo-variable

SUBSTR(s,f,n) and SUBSTR(s,f)

Abbreviations

CHAR for CHARACTER
DCL for DECLARE
EXT for EXTERNAL
GO for GOTO
INIT for INITIAL
OTHER for OTHERWISE
PROC for PROCEDURE
READ for READONLY
VAR for VARYING

Comments

```
/* text */
/** statement comment */
/*+ statement comment */
/** end statement comment */
/*: pseudo-comment */
/*i pseudo-comment */
```

I.B Reserved Words

The following *keywords* of PL/CS are *reserved* and cannot be used as names in a PL/CS program:

ALL	FIXED	PUT
ASSERT	FLOAT	READ
BIT	FOR	READONLY
BY	GET	RETURN
CALL	GO	RETURNS
CHAR	GOTO	SELECT
CHARACTER	IF	SKIP
DATA	INIT	SOME
DATAEND	INITIAL	STATIC
DCL	LEAVE	THEN
DECLARE	LIST	TO
DO	MAIN	UNTIL
EDIT	ON	VAR
ELSE	OPTIONS	VARYING
END	OTHER	WHEN
ENDFILE	OTHERWISE	WHILE
EXT	PROC	
EXTERNAL	PROCEDURE	

The following keywords are *reserved in PL/C* and cannot be used as names in a PL/C program. Their use will make your PL/CS program incompatible with PL/C and PL/I:

ALLOCATE	EXIT	PROC
BEGIN	FLOW	PROCEDURE
BY	FORMAT	PUT
CALL	FREE	READ
CHECK	GET	RETURN
CLOSE	GO	REVERT
DCL	GOTO	REWRITE
DECLARE	IF	SIGNAL
DELETE	NO	STOP
DO	NOCHECK	THEN
ELSE	NOFLOW	TO
END	ON	WHILE
ENTRY	OPEN	WRITE

I.C Built-in Functions

The following is a complete list of the built-in functions that are included in PL/CS. This is only a subset of the PL/C built-in functions, which is a subset of the PL/I built-in functions. A complete list for PL/C and PL/I is given in the PL/C User's Guide and the PL/I(F) Language Reference Manual, respectively.

In the descriptions below:

x	represents an arithmetic expression
i	represents an integer constant
f, n	represent integer-valued arithmetic expressions
s	represents a string expression

ABS(x)
: The result is the absolute value of x.

ALL(bit array)
: The result is a bit value obtained by "and-ing" (as in the operator "&") all the bit values of the array together.

ANY(bit array)
: The result is a bit value obtained by "or-ing" (as in the operator "|") all the bit values in the array together.

ATAN(x)
: The result is the arctangent of x. x is expressed in radians.

ATAN(x1,x2)
: The result is the arctangent of x1/x2. x1 and x2 are expressed in radians and must not both be zero.

ATAND(x1,x2)
: The result is the arctangent of x1/x2. x1 and x2 are expressed in degrees and must not both be zero.

ATANH(x)
: The result is the hyperbolic tangent of x. ABS(x) must be greater than or equal to 1.

CEIL(x)
: The result is the smallest integer that is greater than or equal to x. For example, CEIL(3.5) is 4 and

CEIL(-3.5) is -3.

COS(x)

The result is the cosine of x. x is expressed in radians.

COSD(x)

The result is the cosine of x. x is expressed in degrees.

COSH(x)

The result is the hyperbolic cosine of x.

DATE

The result is a character string of length 6, with the form yymmdd. yy is the current year, mm the current month, and dd the current day. For example, November 13, 1932 is 321113.

DIM(array,i)

The result is an integer giving the "extent" of the ith dimension of the array. The extent is equal to the upper bound minus the lower bound, plus 1.

ERF(x)

The result is 2/SQRT(PI) multiplied by the definite integral from 0 to x of e**(-t^2)dt.

ERFC(x)

The result is 1-ERF(x).

EXP(x)

The result is e**x, where e is the base of the natural logarithm system.

FIXED(x)

The result is the argument x in FIXED form.

FLOAT(x)

The result is the argument x in FLOAT form.

FLOOR(x)

The result is the largest integer not greater than x. For example, FLOOR(3.5) is 3 and FLOOR(-3.5) is -4.

HBOUND(array,i)

The result is the upper bound of the ith dimension of the array.

HIGH(i)

The result is a character string of length i, each character of which is the highest character in the collating sequence.

INDEX(s1,s2)
The result is an integer indicating the leftmost position in s1 where s2 begins. If s2 does not appear as a substring of s1, the result is 0.

LBOUND(array,i)
The result is the lower bound of the ith dimension of the array.

LENGTH(s)
The result is an integer giving the length of string s.

LOG(x)
The result is the natural logarithm of x. x must be greater than 0.

LOG10(x)
The result is the common logarithm of x (base 10 log). X must be greater than 0.

LOG2(x)
The result is the logarithm to the base 2 of x. x must be greater than 0.

LOW(i)
The result is a character string of length i, each character of which is the lowest character in the collating sequence.

MAX(x1,x2,...,xn)
The result is the maximum value of the arguments. If all the arguments are FIXED the result is FIXED. If any argument is FLOAT the result is FLOAT.

MIN(x1,x2,...,xn)
The result is the minimum value of the arguments. If all the arguments are FIXED the result is FIXED. If any argument is FLOAT the result is FLOAT.

MOD(x1,x2)
The result is the remainder when dividing x1 by x2. If x1 and x2 have different signs, the operation is performed on their absolute values, and the result is then ABS(x2)-remainder. For example, MOD(29,6) is 5, while MOD(-29,6) is 1.

PROD(array)
The result is the product of all the elements of the array.

RAND(x)

The result is the next value in a sequence of pseudo-random numbers of which x was the previous value. The method used is that of Coveyou and Macpherson, given in ACM Journal 14 (1967) pgs 100-119. x should be FLOAT. It must be in the range 0<x<1. The initial value of x should have nine significant digits, and be odd; this tends to increase the period of the sequence. RAND is almost always used in an assignment statement of the form:

```
v = RAND(v);
```

where v is a FLOAT variable.

REPEAT(s,i)

The result is s concatenated with itself i times. For example, REPEAT('ABC',2) is 'ABCABCABC'.

ROUND(x,n)

If x is FIXED, if n<0, the result is (x*10**n)/10**n; otherwise x is returned unchanged.
If x is FLOAT, n is ignored and the rightmost bit of the internal binary representation of x is set to 1.

SIGN(x)

The result is 1 if x>0, 0 if x=0, and -1 if x<0.

SIN(x)

The result is the sine of x. x is expressed in radians.

SIND(x)

The result is the sine of x. x is expressed in degrees.

SINH(x)

The result is the hyperbolic sine of x.

SQRT(x)

The result is the square root of x. x must be greater than or equal to 0.

STRING(string array)

The result is the string resulting from concatenating all the elements of the array together.

SUBSTR(s,f,n)

The result is the substring of s consisting of n characters whose leftmost character is in position f. The resulting length is n.

SUBSTR(s,f)
The result is the substring of s starting with position f and continuing to the last character of s. The resulting length is LENGTH(s)-f+1.

SUM(array)
The result is the sum of all the elements of the array.

TAN(x)
The result is the tangent of x. x is expressed in radians.

TAND(x)
The result is the tangent of x. x is expressed in radians.

TANH(x)
The result is the hyperbolic tangent of x.

TIME
The result is a character string of length 9 giving the current time of day, Its form is hhmmssttt, where hh is the current hour of the day, mm is the number of minutes, ss the number of seconds, and ttt the number of milliseconds in machine-dependent increments.

TRANSLATE(s1,s2,s3)
The result is a string identical to s1, except that any character of s1 which is also in s3 is replaced by the corresponding character in s2. Thus if character i of s1 is the same as character j of s3, then character i of s1 is replaced by character j of s2. For example, the result of

```
TRANSLATE('XYZW', 'ABCD', 'VWXY')
```

is the string 'CDZB'.

TRUNC(x)
If x<0 the result is CEIL(x); if x>0 the result is FLOOR(x).

VERIFY(s1,s2) character in s1 which is not in s2.
The result is an integer which indicates the position of the first if all characters are in s2, the result is 0. For example:

```
VERIFY('JJBC', 'J') yields 3
VERIFY('TTBC', 'ABCT') yields 0.
```

I.D PL/CS Options

Options may be specified on the *PL/CS card or on any *PROCESS card. Options may be given in any combination, in any order, separated by blanks and/or commas. They may be continued onto a continuation card with * in column 1 and columns 2-3 blank. But an individual option may not be split over a card boundary, and may not have blanks interspersed (must be dense). Note that a *PROCESS card immediately following a *PL/CS or a *PROCESS card will also be treated as a continuation card. Options may be abbreviated or misspelled; only a few key letters are significant, as indicated in the listing of options below. The prefix letters N or NO designate negated options. Certain options can only be given on the *PL/CS card, as noted.

Options on the *PL/CS card, and the default values for options not specified, are in effect throughout the program, except as temporarily overridden on a *PROCESS card. *PROCESS options apply only to the one external procedure following that card. After each external procedure, options are reset to the "global" *PL/CS and default values.

In the listing below the normal default value of each option is underlined, but these choices are easily changed by each installation, and yours may be different from what is shown here. In addition, each installation can override options so that user specification of such options is ineffective.

CMNTS, CMNTS=(n1,n2,...), NOCMNTS, C

Convert specified classes of pseudo-comments to source text. Comments beginning with : are converted. If parameter(s) are given (1<=ni<=7) comments beginning with ni are also converted.

CMPRS, NOCMPRS, CP

Source listing to be given in compressed form. That means that certain page ejects are replaced by 3 line skips.

CTIME=(m,s.h), CT (on *PL/CS only)

Time limit for compilation:

m is minutes; assumed 0 if omitted
s is seconds; assumed 0 if omitted
h is hundredths of seconds; assumed 0 if omitted.

DUMP, DUMP=(d1,d2,...), NODUMP, D (on *PL/CS only)

Produce post-mortem dump.

Dump options are:

BLOCKS, B
Traceback of blocks active at termination.

SCALARS, S
Final values of scalar variables in active blocks. (Implies B.)

ARRAYS, A
Final values of arrays in active blocks.

```
            (Implies S and B.)
        FLOW, F
            History of last 18 transfers of control.
        LABELS, L
            List of labels with frequency of encounter.
        ENTRIES, E
            List of entry-names with frequency of call.
        REPORT, R
            Statistics on run (time, core usage,
            auxiliary I/O operations, etc.)
        UNREAD, U
            List of first 5 or fewer unread data cards.
        DFLTS, D
            Equivalent to specifying the installation
            defaults.
        Depth
            An integer giving limit on number of active
            blocks for B, S and A dump options.  If 0 is
            given, depth is unlimited.
    Supplied default DUMP options are (B,S,F,L,E,R,U,0).

DUMPE, DUMPE=(d1,d2,...), NODUMPE, DE (on *PL/CS only)
    Produce post-mortem dump only if error was encountered
    during execution.
    Supplied default DUMPE options are (B,S,F,L,E,R,U,0).

DUMPT, DUMPT=(d1,d2,...), NODUMPT, DT (on *PL/CS only)
    Produce post-mortem dump only if execution was terminated
    by an error.
    Supplied default DUMPT options are (B,S,F,L,E,R,U,0).

DUMPS, NODUMPS, DS (on *PL/CS only)
    Specifies all three of DUMP, DUMPE and DUMPT.

ETIME=(m,s.h), ET (on *PL/CS only)
    Time limit for execution:
        m is minutes; assumed 0 if omitted
        s is seconds; assumed 0 if omitted
        h is hundredths of seconds; assumed 0 if omitted.

ERRORS=(c,r), E (on *PL/CS only)
    Suppress execution if c or more compile errors.
        If c=0 suppress execution unconditionally.
    Terminate execution after r runtime errors.
        If r=0 there is no limit on runtime errors.
    Supplied default c=50, r=50.

FLAGS, FLAGE, FLAGW, FS, FE, FW
    FLAGS prints only S and T level messages
    FLAGE prints E, S and T level messages
    FLAGW prints W, E, S and T level messages.

HDRPG, NOHDRPG, H (on *PL/CS only)
    Print header/separator page before program.
```

ID='name', I (on *PL/CS only)
    Program identification name (20 characters maximum).
    Supplied default name = '*** NO ID ***'

INDENT=n, IN
    Number of columns for each level of indentation on source listing. Supplied default n=4.

LINES=n, L (on *PL/CS only)
    Maximum number of lines to be printed.
    Supplied default n=2000.

LINECT=n, LC
    Number of lines per page on source listing. Supplied default n=60.

LIST, <u>NOLIST</u>, LS (on *PL/CS only)
    Print list of source card images (separate from regular source program listing).

<u>MONITOR</u>, NOMONITOR, M
MONITOR=(d1,d2,..), NOMONITOR=(d1,d2,..)
    The MONITOR option specifies that the error message should be printed when the monitored condition arises. NOMONITOR suppresses the message. The MONITOR options are the following:
        ASSERT, A
            Monitor truth of ASSERT statements.
        BNDRY, B
            Limit string constants by card boundary.
        CONV, C (on *PL/CS only)
            Monitor automatic arithmetic/string conversion.
        DFLTS, D (on *PL/CS only)
            Equivalent to specifying the installation defaults.
        UDEF, U (on *PL/CS only)
            Monitor use of uninitialized variables.
    On a *PROCESS card, the options listed after MONITOR are turned on; the options listed after NOMONITOR are turned off. MONITOR or NOMONITOR without options is assumed to apply to all options. On a *PL/CS card the options listed after MONITOR are turned on; all others are turned off.

MSGCOL=n, MS
    Set column on source listing for error messges and ordinary comments. Supplied default is MSGCOL=70.

<u>OPLIST</u>, NOOPLIST, O
    Print list of options in effect.

PAGES=n, P (on *PL/CS only)
    Maximum number of pages to be printed.
    Supplied default n=30.

SORMGIN=(s,e), SM
    Establish source card margins:
        s is first column scanned; supplied default s=2.
        e is last column scanned; supplied default e=72.

<u>SOURCE</u>, NOSOURCE, S
    Print source program listing.

TABSIZE=n, TS (on *PL/CS only)
    Determines amount of PL/CS region allocated to symbol table. n is given in fullwords. Supplied default is 1/2 of usable area, up to 32768 fullwords. This option is used to allow somewhat larger programs to fit in given region size by changing the internal use of space. Base n on information given in post-mortem dump.

TIME=(m,s.h), T (on *PL/CS only)
    Time limit (compilation + execution):
        m is minutes: assumed 0 if omitted
        s is seconds: assumed 0 if omitted
        h is hundredths of seconds; assumed 0 if omitted.
    Supplied default is TIME=(0,15.00).

    Note there are three different ways of limiting the processing time:
        total job time (TIME option)
        compilation time (CTIME option)
        execution time (ETIME option).

I.E Card Formats

For all types of cards the contents of columns 1 and 2 may be significant to the "operating system" and cause a card to be intercepted and never reach PL/CS. The characters // in columns 1 and 2 are significant to most IBM systems, and the characters /* are significant to some. Both combinations should be avoided in 1-2 of all cards (data cards as well as program). A common error is to begin a comment in column 1. For many systems a card with /* in columns 1 and 2 will end the program. (This is ridiculous, but there is nothing PL/CS can do about it. The card is intercepted by the operating system and never reaches PL/CS.) this is the reason we avoid column 1 for all program cards.

If a card with // in 1-2 reaches PL/CS:

1. If 3-80 are blank it is treated as an end-of-file and it terminates the program. PL/CS expects either a *PL/CS card to begin a new program or to have the job ended by the operating system. Any number of consecutive // cards with 3-80 blank are equivalent to one.

2. If 3-80 are not blank the entire card is ignored.

I.E.1 Control Cards

Control cards have * in column 1. (Some installations may use another character instead of *.) The control keyword -- PL/CS, PROCESS or DATA -- begins in 2. The continuation of a control card has * in 1 and 2-3 blank. Note that a *PROCESS card immediately following a *PL/CS or *PROCESS card will also be treated as a continuation card. Control cards are not affected by SORMGIN. Options on control cards can be in any order, separated by blanks and/or commas, but can not be split over a card boundary or have blanks interspersed.

I.E.2 Program Cards

1. The default card field for source statements is columns 2-72. The contents of columns 73-80 are ignored, but appear on the source card listing.

1a. Columns to the right of the right margin (default 73-80) can be used for identification and numbering. A four-character abbreviation of the program name can be punched in 73-76, and automatically duplicated from one card to the next. Cards should be serially numbered in 77-80 with initial numbers in intervals of ten or more to leave room for later insertions. For example, a sorting program might be identified and initially numbered:

SORT0010

```
SORT0020
SORT0030
...
```

(Card numbering may seem unnecessary until you or the computer operator drop one of your decks.)

2. The default source card format can be altered by specifying the SORMGIN option. The form is:

```
SORMGIN=(s,e)
     where:   s is the leftmost column to be included
              e is the rightmost column to be included.
```

3. When the BNDRY option is in effect (usually the default), PL/CS does not permit any element to be split over a card boundary. That is, keywords, identifiers, constants and comments cannot start on one card and continue on the next. This limits the length of literals, (string constants) and means that for long comments each card must be a separate comment.

When NOBNDRY is specified, literals may be continued over a card boundary (as in PL/I). The maximum length of a literal is then 256 characters. Note that the card boundary is as defined by the SORMGIN option and not the physical card boundary. For example, with the default SORMGIN of (2,72) column 2 directly follows 72 of the previous card -- no blank is supplied. Note also that NOBNDRY applies only to literals. In PL/CS you still cannot continue a keyword, a name, an arithmetic constant, or a comment over a card boundary.

I.E.3 Data Cards

The card field for data cards is always 1 to 80. Data cards are not affected by the SORMGIN or BNDRY options. Data cards are considered to be a continuous stream of characters and the card boundary is of no significance. That is, column 1 of a card directly follows 80 of the previous card, and any element may be continued over a card boundary.

Part II

Entry Forms for PL/CS

PL/CS is a remarkably tolerant processor. It will attempt to repair and execute almost anything that is submitted as a source program. The purpose of this error-repair characteristic (which is highly unusual among programming language processors) is to overcome the occasional oversights and keypunching mistakes that afflict most programmers -- especially neophytes, but even experienced programmers as well. The idea is that by running a repaired program, some additional information can be provided to the user that may help in testing and debugging the program.

While PL/CS's repair capability was originally intended to overcome unintentional errors, it has the interesting and convenient characteristic of granting the programmer considerable freedom to abbreviate the statements of his program -- in effect, to _deliberately make errors_ by omitting various redundant words and punctuation marks. The subset of PL/I used in PL/CS is highly redundant, and the processor can correctly interpret your intent in spite of substantial omissions. This way you can significantly reduce the amount of keypunching required for a given program. But note that as you reduce the redundancy of the program you present to PL/CS you also reduce its power to repair any real (unintentional) errors you may have committed. It is probably a good idea to initially use the full "display form" of the language, as shown in the text, at least until you are very familiar with it. Then you can develop some abbreviated "entry form" that is a suitable personal compromise between compactness of input, readability of input, and protective redundancy.

You can develop your own entry form by experimenting to see just "what you can get away with", and still have PL/CS correctly interpret your statements. The entry forms described below are simply suggestions, and do not have to be followed rigidly. In effect, each entry form is really a _different programming language_. But it is a programming language carefully "designed" so that the PL/CS processor can correctly expand the abbreviated input into full, PL/I-compatible display form.

If you use an abbreviated entry form you should be aware of the scheme by which PL/CS classifies the _severity of errors_. There are four levels of severity:

Warnings (W level). A possibly suspicious construction, but the program is not altered by PL/CS.

Errors (E level). Input requires alteration by PL/CS to achieve correct and complete display form, but the repair is fairly obvious and is considered very likely to achieve what the programmer intended.

Serious Errors (S level). The program has been altered by PL/CS to allow it to execute, but the necessary change was sufficiently severe that it is unlikely to achieve what the programmer intended.

Terminal Errors (T level). The program has been altered by PL/CS, but the prospects of obtaining useful information from execution are so dubious that execution is not even attempted.

You can control the severity level of the error messages that will be printed (see Section I.D). For example, if you run using the option FLAGS, only S and T level messages will be printed on the source listing. If you want to see E level messages you must run with the FLAGE option. The _normal default message level_ is FLAGS -- so if you do not specify any message level option, only S and T level messages will appear. This provides a practical limit on the extent of abbreviation of an entry form -- the _abbreviation should not result in an S or T level error_. This means that normally the source listing will show only the "corrected" form of the program, and not be marred by error messages. However, even when messages are not printed, the code letter (W, E, S or T) is printed in column 5 of the source listing to show that messages were generated for that statement. (You will have to rerun with a different message-level option to see the messages.) There will also be a code "G" printed in column 6 for any statement that is _generated_ entirely by PL/CS -- that is, for a statement that has no counterpart in the entry form.

One final remark is necessary. The wisdom of abbreviation and reliance on language defaults and processor repairs is still undecided. Part II is intended simply to organize a phenomenon you would surely have discovered for yourself, and is not necessarily a recommendation of the practice. As a matter of personal preference, we generally recommend an entry form at roughly level II, described below. This is far from minimal, but it provides easily readable input cards, and substantial recovery from accidental errors.

<u>Level I</u> - PL/I Abbreviations and Defaults

This level represents abbreviations that even PL/I will tolerate. These abbreviations are defined as part of PL/I, and hence are not considered errors at all. The "defaults" of a programming language are just the specification of what the language assumes in the absence of explicit programmer instructions.

1. Keyword abbreviations. Certain keywords can be abbreviated, as follows:

   ```
   CHAR  for CHARACTER
   DCL   for DECLARE
   EXT   for EXTERNAL
   GO    for GOTO
   INIT  for INITIAL
   OTHER for OTHERWISE
   PROC  for PROCEDURE
   READ  for READONLY (not PL/I attribute)
   VAR   for VARYING
   ```

   These are the <u>only</u> abbreviations allowed, even in PL/CS. While PL/CS has a limited ability to correct spelling mistakes in certain keywords, this does not generally allow you to invent your own abbreviations of keywords.

2. Implicit Declaration of Variables. PL/I allows a variable to be used without explicit declaration, and assigns type attributes that depend upon the first letter of the variable name. PL/CS also allows the omission of declarations, but differs in two respects:

   a. the default type is FLOAT, regardless of the name

   b. the declaration is printed, just as if it had been given explicitly in the input.

3. Loop and SELECT-names. PL/I allows, but does not require, names on DO and SELECT statements. PL/CS requires names for these statements, but will generate them (from the statement number) when they are omitted.

4. Implicit and Unnamed ENDs. PL/I allows a name suffix on END if the corresponding opening statement was named, but does not require the suffix. PL/CS requires the name suffix on END and will supply it if it is omitted.

   PL/I allows a named END to imply the END of inner loops. PL/CS allows this, but actually prints the ENDs that are generated as a result (marked with a G in column 6).

5. Increment "BY 1" in an Indexed Loop. PL/I assumes "BY 1" if the BY-phrase is omitted. PL/CS makes the same assumption, but actually prints the assumed "BY 1". This applies both to indexed loops and indexed assertions in PL/CS.

Level II -- Redundant Keywords in PL/CS

1. Since the main procedure is always the first procedure OPTIONS(MAIN) can be omitted.

2. In the declaration of a variable:

```
CHAR (n)  implies  CHARACTER (n) VARYING
BIT       implies  BIT (1)
```

In the declaration of a parameter:

```
CHAR  implies  CHARACTER (*) VARYING
BIT   implies  BIT (*)
```

In the RETURNS phrase of a function procedure:

```
FLOAT implies  RETURNS (FLOAT)
FIXED implies  RETURNS (FIXED)
CHAR  implies  RETURNS (CHARACTER (256) VARYING)
BIT   implies  RETURNS (BIT (1))
```

3.
```
WHILE  implies  DO WHILE
UNTIL  implies  DO UNTIL
ALL    implies  FOR ALL
SOME   implies  FOR SOME
SKIP   implies  PUT SKIP
```

4. In the absence of DATA or EDIT in a GET or PUT statement, LIST format input or output is assumed.

5. ON ENDFILE GOTO DATAEND; is implied by the presence of DATAEND:; in the procedure.

6. END proc-name; implies:

```
RETURN;
END proc-name;
```

Level III - Redundant Punctuation in PL/CS

Almost all of the punctuation in PL/CS is actually redundant, and can be omitted. However, in general, the punctuation is quite useful in the correction of accidental errors. When punctuation is omitted, an otherwise correct program will be properly expanded to full display form, but an accidental error can cause very confusing "repairs". Punctuation is also very helpful in making the raw input readable, so we do not recommend abbreviation to this level unless you are very confident of your command of PL/I.

1. A semi-colons can be omitted.

2. Parentheses enclosing lists, expressions and conditions can be omitted. Parentheses that specify precedence in expressions and conditions, and that define repetition in format or initial value lists must, of course, be given.

3. Commas between elements of a list can be omitted.

4. Colons after names (procedures, functions, loops, select, target-label, DATAEND) can be omitted. Colons to separate upper and lower bounds in the declaration of an array must be given.

Higher Levels

Even more severe abbreviation than that described above is possible, but is really more of a game, to see what you can get away with, than a practical tool to reduce the volume and effort of input. We decline to describe further abbreviation so as not to spoil the game, and to avoid implying endorsement of extreme abbreviation.

Part III

Source Language Specifications

Release 2

The following is a complete list of PL/CS features. Except as noted below, PL/CS-2 syntax and semantics are identical to the corresponding PL/C statement.

Statements

ASSERT

1. Form is: ASSERT (condition) [quantifier];
2. Quantifier is either:
   FOR ALL index-var = exp1 TO exp2 BY exp3;
   FOR SOME index-var = exp1 TO exp2 BY exp3;
3. If satisfied, ASSERT has no side-effects; if not satisfied, ASSERT prints a message. Blind variables and labels generated do not appear in PM dump.
4. Expansion is controlled by the ASSERT sub-option of MONITOR.

Assignment

1. No multiple left-hand-sides.
2. In array assignment, right-hand-side must be an array or a constant.
3. In assignments to a bit variable, the right-hand-side must be enclosed in parentheses.
4. No assignments to READONLY variables or parameters.

CALL

1. Neither labels nor entry-names are allowed as arguments.
2. Types of arguments and parameters must match exactly.
3. Arguments cannot be READONLY parameters or variables.
4. Arguments cannot be control variables of a loop.

DATAEND

1. Form is DATAEND:;
2. Can appear at most once per procedure.

3. Must be positioned after last GET statement in procedure.
4. Must be at "top-level" -- cannot be in compound statement or loop, or in IF or SELECT unit.

DECLARE
1. Explicit declaration of variables and parameters is required, and must be positioned at the beginning of the procedure.
2. Variables and parameters must be given in separate declarations.
3. Form is: DECLARE (ident-list) attributes;
4. Array dimensions are as in PL/C.
5. Attributes for variables:
   a. One of following is required:
      FLOAT, FIXED, CHARACTER (length) VARYING, BIT (1)
   b. Optional:
      INITIAL (list), STATIC, EXTERNAL, READONLY
   c. Required order:
      type INITIAL STATIC READONLY
      EXTERNAL
   d. Combinations:
      READONLY requires EXTERNAL or INITIAL
6. Attributes for parameters:
   a. One of the following is required:
      FLOAT, FIXED, CHARACTER (*) VARYING, BIT (*)
   b. Optional:
      READONLY
   c. Required order:
      type READONLY
7. FLOAT implies FLOAT DECIMAL(16),
   FIXED implies FIXED DECIMAL(15,0)

DO
1. Compound statement as in PL/C, but no label.
2. DO WHILE as in PL/C, with single label required.
3. DO UNTIL (condition); with single label required.
4. DO index-var = exp1 TO exp2 BY exp3;
   a. Single label required.
   B. Form shown above is required:
      TO-BY phrases must be in the order shown.
   C. Index-var must be numeric.
   D. Index-var cannot be a READONLY variable or parameter.
   E. Index-var cannot be EXTERNAL.
   F. Index-var and all variables in exp1, exp2, and exp3 (that is, the control variables) are READONLY in the body of the loop. They cannot be the target of an assignment, cannot appear in the variable list of a GET statement, CALL statement, or be the index-var of another (nested) loop.
   G. The value of the index-var after normal termination of the loop is unitialized.
   H. Exp3 must have a non-zero value.

END

1. All ENDs are explicit.
2. All ENDs except for compound statements must be named.

GET

1. Only the LIST, DATA and EDIT options are are allowed.
3. No DO iteration in the variable list.
4. Cannot appear after DATAEND.
5. READONLY variables and parameters cannot appear in the variable list.

GOTO

1. Only forward references are allowed.
2. Target must be a labeled null statement.
3. Target can be in loop or compound statement only if all references are also in that loop or compound statement.

IF

1. Form is: IF (condition) THEN s1; [ELSE s2;]
2. Neither s1 nor s2 can be null.

LEAVE

1. Form is: LEAVE loop-name:
2. Can only appear in the body of the named loop.
3. Blind labels generated do not appear in PM dump.

Null

1. Form is: label:;
2. Used only as a target for a GOTO statement.

ON

1. Form is: ON ENDFILE GOTO DATAEND;
2. Can appear at most once per procedure.
3. Must be positioned immediately after declarations.

PROCEDURE

1. MAIN procedure must be the first procedure.
2. All procedures are external.
3. All parameters and variables must be explicitly declared at the beginning of the procedure.
4. Types of arguments and parameters must match exactly.
5. Lengths and dimensions are dynamic as in PL/C.
6. Function procedures have no side-effects:
   - a. All parameters are READONLY.
   - B. STATIC and EXTERNAL variables are READONLY.
   - C. Body cannot contain CALL, GET or PUT.
7. Function procedures must have at least one parameter.
8. Character-string function is specified:
   RETURNS(CHARACTER(256)VARYING)
9. Bit-string function is specified:
   RETURNS(BIT(1))
10. Single entry-name is required.
11. Explicit RETURN before the END is required.

PUT

1. Only the SKIP, LIST, DATA, and EDIT options are allowed.
2. No DO iteration in the variable list.

RETURN	1.	Explicit RETURN before the END is required.
SELECT	1.	Form is: select-name: SELECT;

```
                    WHEN (condition1) s1;
                    WHEN (condition2) s2;
                        ...
                    OTHERWISE sn;
                    END select-name;
```

2. At least one WHEN must be given.
3. OTHERWISE is required.

Comments

1. Procedure comments (/*) print at left.
2. Ordinary comments (/*) print in MSGCOL;
   They can only follow statements.
3. Declaration comments (/*) print in MSGCOL;
   Can follow identifiers.
4. Pseudo-comments (/*: or /*i i=1-7) are included;
   Depending on CMNTS option, they are converted to ordinary comments or normal source text (without comment delimiters).
5. Statement comments (/** or /*+);
   Printed in source text as a DO statement would be.
   /*+ implies (but does not generate) a closure of the last /**.

Miscellaneous

1. *OPTIONS card is not included.
2. Macro facility is not included.
3. *INCLUDE facility is not included.
4. Trace is deferred.

Abbreviations

CHAR	CHARACTER
DCL	DECLARE
EXT	EXTERNAL
GO	GOTO
INIT	INITIAL
OTHER	OTHERWISE
PROC	PROCEDURE
READ	READONLY
VAR	VARYING

Reserved Words

ALL	DCL	FIXED	LEAVE	PUT	THEN
ASSERT	DECLARE	FLOAT	LIST	READ	TO
BIT	DO	FOR	MAIN	READONLY	UNTIL

BY	EDIT	GET	ON	RETURN	VAR
CALL	ELSE	GO	OPTIONS	RETURNS	VARYING
CHAR	END	GOTO	OTHER	SELECT	WHEN
CHARACTER	ENDFILE	IF	OTHERWISE	SKIP	WHILE
DATA	EXT	INIT	PROC	SOME	
DATAEND	EXTERNAL	INITIAL	PROCEDURE	STATIC	

Built-in Functions

ABS	COSH	FLOOR	LOG2	ROUND	SUM
ALL	DATE	HBOUND	LOW	SIGN	TAN
ANY	DIM	HIGH	MAX	SIN	TAND
ATAN	ERF	INDEX	MIN	SIND	TANH
ATAND	ERFC	LBOUND	MOD	SINH	TIME
ATANH	EXP	LENGTH	PROD	SQRT	TRANSLATE
CEIL	FIXED	LOG	RAND	STRING	TRUNC
COS	FLOAT	LOG10	REPEAT	SUBSTR	VERIFY
COSD					

Options

CMNTS	DUMPT	HDRPG	MONITOR	MSGCOL	TIME
CMPRS	DUMPS	ID	ASSRT	OPLIST	
CTIME	ETIME	INDENT	BNDRY	PAGES	
DUMP	ERRORS	LINES	CONV	SORMGIN	
DUMPE	FLAGS	LINECT	DFLTS	SOURCE	
		LIST	UDEF	TABSIZE	