IPIC
6537
Date:

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

PL/I BULLETIN NO. 3

March 1967

CONTENTS

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

The PL/I Bulletin is sponsored by Working Group 4 (WG4) of the Special Interest Group on Programming Languages (SIGPLAN) of the Los Angeles Chapter of the Association for Computing Machinery.

The opinions and statements expressed by contributors to this bulletin do not necessarily reflect those of the sponsor, and the sponsor undertakes no responsibility for any action which might arise from such statements. Furthermore, publication of programs or algorithms in this bulletin does not constitute endorsement of their correctness or accuracy. The sponsor does not retain copyright authority on material published here, except in the case of material produced by WG4 itself. Permission to reproduce any contribution should be obtained directly from the authors.

Correspondence and contributions should be addressed to:

R. N. Southworth (Editor, PL/I Bulletin)
c/o Logicon, Inc., 205 Avenue I
Redondo Beach, California 90277

PB3.0 EDITOR'S NOTES

PB3.0.1 On Contributions

The PL/I Bulletin is intended to be an informal publication for the interchange of information and opinions relating to PL/I. Correspondence and other contributions from the general readership are solicited. The frequency of publication has been and will continue to be adjusted to the rate of input.

PB3.0.2 Some Changes

The "NEWS ITEMS" section of this bulletin will be de-emphasized because of the excellent reporting of PL/I news in SICPLAN notices. The "CIRCULATION LIST" section will be dropped because of the change in the method of circulation of the PL/I Bulletin (see SICPLAN notices).

Sperry Rand AG
Bärengasse 29
8001 Zürich
January 27, 1967

PB3.2 CORRESPONDENCE

PB3.2.1 ECMA/TC 10 Activities

I am so sorry for the delay in answering your letter dated 9/28/66 to Mr. Cormack.

Please find enclosed a short report on the ECMA/TC 10 activity. I hope it properly represents the past work and the ideas of TC 10. Further included are copies of 6 pages extracted from ECMA document ECMA/TC 10/67/2 (title, pages i, ii, 110, 111, 112). I think those pages summarize the complete document.

The ECMA secretariate will send you the complete document ECMA/TC 10/67/2 and other ECMA documents that might be of interest to you.

Sincerely yours,
Dr. W. Schöniger
Chairman, ECMA/TC 10

(Editor's Note: The report mentioned by Dr. Schöniger follows below. The full text of ECMA/TC 10/67/2 is to be printed in PB4.)

Results and Present Status of Work within TC 10

1) Concentration on Subsets at Present

In ECMA/TC 10 standardization work is concentrated on subsets of PL/I at present. There are no fundamental objections against the full language. However, TC 10 members feel that especially those parts of the language, that are beyond the D-level are comparatively new features and need careful checking and possibly clarification. Users' and implementors' feedback is necessary especially for those parts to determine if and how they should be included into the language.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

2) Minimum PL/I Subset for Scientific + Commercial Users

ECMA/TC 10 has now completed specifications for a minimum PL/I subset for scientific + commercial users, a complete document is available. However, this subset is only supported by three companies.

3) PL/I Subset Approximately at IBM D-level

ECMA/TC 10 is at present specifying a PL/I subset approximately at the IBM D-level. - This subset is supported by a large majority of companies, represented in ECMA/TC 10.

4) Formal Definition of PL/I

TC 10 members are trying to get enough knowledge to decide on the suitability of formal methods for a rigorous and clear definition of the full language and of subsets.

5) Full language

During this year work on the full language will be started again.

6) Character Set

Standardization effort within ECMA/TC 10 on PL/I will be based on the 60-character-language as described in the IBM language specification. The 48-character-language will no longer be considered for standardization.

The character set for PL/I should definitely lie in columns 2-5 of the ECMA 7-bit-code standard.

Logical OR should be represented by ! .
Logical NOT should be represented by ^ .

Only the letters A through Z should be considered alphabetic.

7) Collating Sequence

The collating sequence should be implementor defined. However, the collating sequence implied by the ECMA 7-bit-code is recommended.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

PB3.2.2 Comments on PL/I

February 19, 1967

From the experience gained in programming the PL/I program, REFORMAT, I would like to suggest the following additions and changes to the language.

1) DO I = A should be allowed, where A is an array, and I is set to each successive value of A.

2) UNTIL, as suggested by W. N. Holmes (PB2.3.5) should be added.

3) UNLESS: not restricted to places where the IF statement is allowed, as implied by Mr. Holmes, but also to be used in:

DO I = 1 TO K UNLESS (J)

(where J may be an expression or variable) to mean if J is true then go to the corresponding END statement (i.e., increment I).

4) RELOCATE should be provided to alter the size of an existing (allocated) block, while not destroying its contents (except of course in the case of REALLOCATE-ing to a smaller block).

5) The data attributes on a PROCEDURE or ENTRY statement should define that PROCEDURE or ENTRY as "RETURNS (data attributes)."

6) DEFAULT: The DEFAULT statement should be included as a way of specifying the default attributes within a block, so that if all (or most) variables within a particular procedure are; e.g., FIXED BINARY STATIC only, they need not be specifically declared such, but rather will default to those attributes.

Beyond the above suggestions, I found PL/I to be a far more powerful language than it appears to be at first glance. One case in point, REFORMAT could not have been written easily (more probably not at all) in FORTRAN, as it involves extensive character- and bit-string manipulation.

Margaret S. Schneider
System Development Corporation

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

PB3.3 WORKING PAPERS

PB3.3.1 A PL/I Reformatter

Margaret S. Schneider

The PL/I programming language allows the programmer an unprecedented degree of flexibility in the way in which he presents his program to the compiler: statements may appear anywhere on a card, may span cards, or a number of statements may appear on one card; comments may be inserted anywhere there is a blank, and keywords may be abbreviated. This flexibility facilitates modification and correction when a program is being written or checked out, however, when the program is finished, the result is a hodge-podge of symbols randomly scattered across the listing, making it difficult to follow the logic of the program.

The purpose of the reformatter is to accept any PL/I program and rewrite it in a canonical form similar to that suggested in PL/I bulletin No. 1. Statements are placed one per line, and indented or outdented to indicate the structure of the program; labels are moved to the left margin to make them easy to find; a standard form of punctuation is used; etc.

General algorithm: Separate program into "tokens," a token being a special character, a PL/I verb, or an alphanumeric token: part of a multi-word verb, a comment in its entirety, or a variable.

Determine the extent of each statement (i.e., where it effectively ends, e.g., after the "THEN" for an IF.....THEN type statement. For each statement determine if it has a pair (two alphabetic tokens in a row,

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

ignoring comments), and/or an equal sign, and depending upon the result of this, find out what type of statement it is: assignment statement, "arbitrary" statement (one not affecting the reformatting of the rest of the program), or a statement that will affect reformatting, such as a DO statement, or an END statement.

Print the statement in the reformatted form, (deleting extra blanks) as determined by previous statements and itself, and return to process the next statement.

Some points of interest which were encountered and taken into consideration while writing the REFORMAT program: comments may be anywhere a blank may be, and not affect the validity of the program. A blank may occur between any two words or symbols, including between the words of multi-word verbs. The only exception to this is between two-symbol operators; e.g., */or ** or > =. PL/I has no reserved words, leading to statements such as:

```
IF IF = THEN THEN PUT LIST (LIST, PUT)
```

The program will be run using itself as data.

Sample Input:

```
    PERIOD:  IF SUBSTR (PROGRAM, IP, 2) = '.' THEN DO;
          CK:  IF IP = BGNTOKN  THEN IF SUBSTR
(PROGRAM, IP+1, 1) > = '0' THEN GO TO TOKEND; ELSE DO:DO; CALL
     ISTOKEN;  ALPHA(T) = '01000'B;
               BGNTOKEN = IP + 1;
                          GO/*HELP*/TO TOKEND; END; ELSE DO;
IF SUBSTR (PROGRAM, BGNTOKEN, 1) > = '0' THEN GO TO TOKEND;
     ELSE DO; CALL ISTOKEN; BGNTOKN = IP; GO TO CK; END;; CALL
  IMAGE; END; END:END;
```

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

Reformatted Output:

```
PERIOD: IF SUBSTR (PROGRAM, IP, 2) = '.'
        THEN DO;
CK:                 IF IP = BGNTOKN
                    THEN IF SUBSTR (PROGRAM, IP + 1, 1) > '0'
                         THEN GO TO TOKEND;
                         ELSE
DO:                           DO;
                                   CALL ISTOKEN;
                                   ALPHA (T) = '01000'B;
                                   BGNTOKN = IP + 1;
                                   GO /*HELP*/TO TOKEND;
                              END;
                    ELSE DO;
                              IF SUBSTR (PROGRAM, BGNTOKN,
                              1) > = '0'
                              THEN GO TO TOKEND;
                              ELSE DO;
                                        CALL ISTOKEN;
                                        BGNTOKN = IP;
                                        GO TO CK;
                                   END;

                              CALL IMAGE;
                         END;
END:          END;
```

(Editor's Note: The full text of the program, REFORMAT, will be published in the next PL/I Bulletin.)

PB3.3.2 A User's Experience with PL/I

A. E. Chapman

"PL/I, PL/I, wherefore art thou PL/I?" This lament comes from a programmer who has suddenly realized that just knowing the PL/I language is not enough, one also needs to know how it has been implemented.[1] This part of it turns out to be almost impossible to find out except by the user's experimentation with PL/I.[2]

---

[1] IBM S/360 Model 50

[2] Only the British knew and they were keeping it to themselves.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

While there were many problems associated with the first release of the PL/I compiler, the second release has turned out to be much cleaner and, in fact, is at the stage of development where Version 1 should have been when it was released. However, this article has relatively little to say about the compiler problems, or even about the language.[3] Instead, I will concentrate on the present implementation of the PL/I and attempt to show why much of the current unhappiness stems from the implementation, not the language.[4,5] Any reference to PL/I will concern itself with Release 2 only.

Program Structure

PL/I programs are structured in the form of blocks, as in ALGOL, except that PL/I takes the concept even further since statements may be blocked (i.e., delimited) by the following:

```
PROG: PROCEDURE;          BEGIN;          DO;
    Statements          Statements      Statements
END PROG;               END;            END;
```

Ths use of the above, plus such features as dynamic storage allocation, external and internal procedures, and embedded declarations can have a considerable effect upon the running time of a PL/I program.

General Structure of a Program

Since the only implementation of PL/I is heavily tied to IBM's OS/360 with its adherent advantages and disadvantages, a good deal of

---

[3] Except to say that while PL/I is more than adequate for most D.P. applications, certain features should be added to this language, such as SORT and table capabilities.

[4] When we later found out how PL/I had been implemented, we almost decided it was not worth going on.

[5] Those theorists who insist that since PL/I is a compromise language it cannot be good will probably disagree.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

overhead has to be built into a PL/I program to allow it to function within this environment.

An executable PL/I program consists of in-line code, calls to OS, plus calls to subroutines (library routines) which in fact do most of the work in a PL/I program. Library routines do such things as type conversion,[6] initialization of variables, comparisons,[7] built-in functions, operations, rectification of a variable address and a host of other operations.

Specifics of a Program

Associated with any PROCEDURE or BEGIN block is a prolog which performs such functions as allocation of storage and initialization of variables, rectification of addresses, activation of ON conditions for the handling of interrupts and the setup of a save area trace for the linking of procedures. Also associated with a PROCEDURE or BEGIN block is an epilog which functions as the reverse of a prolog. The time spent in a prolog can vary anywhere from a microsecond to 500 milliseconds,[8] depending upon what needs to be done before execution of any statements within the block can commence. An epilog similarly takes as much time as its associated prolog. It can thus be seen that the activation of too many PROCEDURE or BEGIN blocks at object-time can add to the running time of a PL/I program by a great deal.

Last, but by no means least, the object time efficiency is affected by the indirectness used by the compiler in the setup of a PL/I program.[9]

---

[6] Binary to float, float to decimal, etc.
[7] If field lengths are unequal.
[8] Model 50 Under OS.
[9] The object-code produced can be both re-entrant and recursive, provided there is no static storage involved.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

Each structure, array (and in many cases a variable) will have a dope vector associated with it which contains information for use by the program at object-time. In addition to dope vectors, many variables have DED's (Data Elements Descriptors)[10] and skeleton vectors associated with them. All of the features mentioned above contain the seeds of object-time inefficiency due to the overhead necessary to access them.

As a matter of fact, less than 1/3 of the overall inefficiency of a program can be attributed to its data organization, the rest being due to the inefficiency of the object-code and the use of library routines. For instance, where six instructions would suffice, PL/I has fifteen; or where an expression or sub-expression could be calculated at compile-time,[11] it is done at object-time. But even this is minutiae compared to the inefficiency produced by the library routines. This is because almost all of these routines are interpretive by nature (and decree). While this makes them fairly easy to produce, checkout and implement, it can readily be seen that run-time efficiency can easily become a laugh. It is the author's opinion that the incorporation of most of the library routines' functions into main-line code,[12] plus the use of static storage as a default would do the most for improving PL/I's run-time.

My PL/I Experiences

Early in the game I discovered that a program that made extensive use of internal and/or external PROCEDURES and BEGIN blocks kept

[10] For use mainly by arithmetic and type conversion library routines.
[11] For example - subscripts.
[12] A minimum improvement of execution time of at least 50% is not at all impossible to contemplate.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

the execution speed of the program so low as to make it virtually useless. To correct this, all procedures (except the main one) had to be incorporated into in-line source code, placed at the end of the main procedure and branched to directly. Returns were made via a label variable.[13] Additionally, all BEGIN-END's except those used with ON-conditions were replaced by DO-END's which performed the same function except that there was no overhead attached to their use.[14] Upon making these changes alone, the program went from a run-time of 68 minutes to one of 32 minutes.

It was during the testing of the above program that several things happened that drove the author literally up a tree. These things have to do with the implementation and its interface with OS/360 and are described in detail below.

1) Embedded Declarations

Since data declarations can be embedded within a sequence of executable statements I did so. It then turned out that the DECLARE's were, themselves, executable in the sense that each time the sequence of flow passed through the declare's, storage was allocated.[15] Thus, if something is placed in the variable declared and the declare is then executed, a further reference to the variable will be meaningless as the contents now could be anything depending upon what that core location is contained in the skeleton vector that points to the variable. In fact, if placed within a DO loop it is possible to keep allocating storage until there is

[13] The label variable having been set up prior to the branch.
[14] The use of BEGIN's is a concession to ALGOL users and is worthless since a DO can perform just the way a BEGIN can and is more valuable.
[15] Provided the storage class is automatic.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

no more core in which case we could get blasted off the machine and get a core dump and message saying there is no more core available.

2) Controlled Storage

Since the use of controlled storage can be quite attractive in programming I decided to use it to simulate a push-down stack by means of the following mechanism:

```
DCL  OUTFIELD (*) CHAR (32) ;
DCL  INFIELD (*) CHAR (32) ;
DCL  A_STACK CHAR (32) CONTROLLED ;
ALLOCATE A_STACK ;

AO1 :
     DO I = 1 TO INDEXI ;
     A_STACK = INFIELD (I) ;
     ALLOCATE A_STACK ;
     END ;

BO1 :  ≷   ≷   ≷

CO1 :
     DO J = 1 TO INDEXI ;
     OUTFIELD (J) = A_STACK ;
     FREE A_STACK ;
     END ;
```

This arrangement worked just fine except that an error occurred and INDEXI was set to an amount at CO1 greater than it was at AO1. This happened between labels BO1 and CO1 with the result that the FREE operation took place 25 times more than the allocate. It might be expected that an error would occur at this time terminating the program. Instead, the program went executing merrily away using both the good and bad data it now had. Tracking this one down was an absolute horror, expecially since the trace

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

feature (check) was not working at this time. It turns out that the ability to FREE a controlled variable more times than it was allocated is perfectly legal.

3) Editing and Pictures

It might be expected that the use of pictures in data declearations ala COBOL would be an advantage to the programmer. However, it turns out that whenever data is placed within such a variable it is edited in, character by character, via the interpretive library routines. This can be horrible if there are many such variables. Thus, the slogan: 'NO GODDAM PICTURES', except where absolutely necessary, of course.

4) Fixed Decimal

A fixed decimal variable (defined as (5, 2)) was added to another fixed decimal variable (defined as (5, 2)) and then printed out. Only the first two places were printed out. In the above two sentences is contained three weeks of agony since it took me that long to figure out what was happening, the why of it I have never figured out, nor has anyone ever been able to tell me. It so happens that three blank spaces are automatically added to the front of all fixed decimal variables. For example, a variable declared as (5, 2) is in reality (8, 2). The result was mass hysteria among programmers because if you want the variable to be printed as it was declared, it must be edited by means of pictures, GET/PUT STRING statement, SUBSTR function, or some other subterfuge.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

5) Interrupts

Since certain types of interrupts can be handled by the program they were duly placed in the source code without any real understanding of what would happen. It turns out that when any interrupt occurs which the program is not set up to handle, processing is halted and a dump of core is taken. This is fine except, for those conditions that are defined, functioning may not be the way the documentation states.[16] As an example of this, take the use of the ON CONVERSION feature. This was used by me to determine if a character data field was numeric or not. For example, the following:

```
DCL  LBLA LABEL ;
DCL  NUM FIXED BINARY (6) ;
DCL  FIELD CHAR (6) ;
ON   CONVERSION BEGIN ;
                FLAG = 1 ;
                GO TO LBLA ;
                END ;

LBLA = AOO1 ;
NUM = FIELD ;
/* NUMERIC */
  ≀    ≀    ≀
AOO1 :
/* NOT NUMERIC */
  ≀    ≀    ≀
```

The above mechanism works fine except for the case where FIELD contains zero's and/or one's except the last character which is B.

The result is that the PL/I library conversion routine thinks this is a binary bit string. This is not strictly legal because a binary bit string should be defined[17] as a string

[16] A better question might be - 'What does function the way the documentation states, that is if you can understand the documentation'.

[17] PL/I Specifications, -4.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

of 0's and/or 1's followed immediately (no spaces) by two quote marks with a B between them. So much for getting tricky, there should be some way of easily determining if a character variable is numeric or alphabetic.

6) Built-In Functions

The utility of the built-in functions: LENGTH, INDEX and SUBSTR, has turned out to be very valuable, but, being very interpretive by nature, the time they take to execute may vary from 3 to 24 milliseconds. It can thus be said that extensive use of these functions is to be frowned upon. The way to get around using these functions is to overlay a character string with a one-dimensional packed character array (e.g., DCL A CHAR (10); DCL B (10) CHAR (1) DEFINED A PACKED;) and then iterate via a DO loop down the array.

7) Input-Output

The input/output features of PL/I are quite versatile. However, one pays for this in the time taken to read or write data. The use of record I/O can speed up operations by 80% or more over GET/PUT edit, and 60% or more over GET/PUT edit for release 1.[18] If a lot of I/O is being done any decrease in speed becomes important, but compared to its other problems a lot can be said for PL/I's I/O operation.

Conclusions

Over the past eleven months I have done extensive work in PL/I

[18] GET/PUT EDIT for release 2 is slower than that for release 1, and it is also 1500 bytes longer.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

ranging from coding programs to running timing, storage, and function tests of PL/I in a variety of modes. In this time I have found PL/I to be an extremely versatile language,[19] even though a good deal of its power must be restricted to get a decrease in run-time. While much of the present implementation must be improved, the language as it now stands is, in my opinion, a success. What is complicated to program in COBOL (i.e., took 5086 lines of source code) was extremely easy in PL/I (i.e., 943 lines of source code). Execution times (both on Mod 50) COBOL E vs. PL/I F - level placed PL/I 20% faster. It took just as long to convert the program from 7010 COBOL to 360 COBOL as it did to write the PL/I version. I am now, and will continue to be, pro PL/I (as a language), but unless the implementation is improved so that certain self imposed restrictions[20] can be removed, my feeling would be scrap it, I dislike getting a taste of freedom, but never being able to get there.

---

[19] After three years of programming in FORTRAN, COBOL or JOVIAL, PL/I is a blessing, albeit heavily disguised.

[20] See attached appendix for a list of restrictions and suggestions for speeding a PL/I program.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

APPENDIX

How to speed up a PL/I Program:

1. Everything used in a PL/I program must be declared (for safety if nothing else).
2. All storage must be declared STATIC.
3. All array dimensions must be declared.
4. Use as few internal and/or external procedures as possible.
5. All 'subroutine functions' (converted internal/external procedures) will use label variables.
6. All subscripting and arithmetic operations will be done by variables declared as FIXED BINARY.
7. The following attributes and operations will be used as little as possible:
   a) VARYING
   b) BIT
   c) CONCATENATION
8. The following built-in functions will be used as little as possible and then with discretion:
   a) SUBSTR
   b) INDEX

   The need for these can be drastically reduced by use of the DEFINE and POSITION (overlay) attributes combined with subscripting.
9. TYPE conversion should be kept to as few operations as possible.
10. Complex IF statements will not be used. For example:

Do not use:

```
IF EX1 OR
     EX2 OR
       EX3 OR
         EX4 THEN ---;
```

Instead, use:

```
IF EX1 THEN --;
IF EX2 THEN --;
IF EX3 THEN --;
IF EX4 THEN --; ELSE --;
```

for they generate less code and execute faster.

11. Comparisons must be done between fields of equal length if at all possible.

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

PB3.4 PROGRAMS

PB3.4.1 A Permutation Algorithm

F. W. Schneider

```
  DECLARE PERM ENTRY((*,*) FIXED BINARY,(*) FIXED BINARY) SETS (1);
PERM:PROCEDURE(A, V);
/*  THIS PROCEDURE HAS TWO ARGUMENTS:
      1) AN ARRAY N! BY N WHICH WILL CONTAIN THE PERMUTATIONS;
      2) AN N-VECTOR CONTAINING THE SYMBOLS TO BE PERMUTED.
    THE PROCEDURE WILL SET THE FIRST ARGUMENT TO CONTAIN THE N!
    PERMUTATIONS OF THE SECOND ARGUMENT.

   THE ALGORITHM IS TAKEN FROM THE LAST EXAMPLE OF THE ARTICLE
       EULER:A GENERALIZATION OF ALGOL, AND ITS FORMAL DEFINITION:
                   PART II    NIKLAUS WIRTH AND HELMUT WEBER
             ( CACM 9 #2 FEBRUARY 1966   PG. 89  )              */
     DECLARE (A(*,*), V(*)) FIXED BINARY;
     I=1; N=DIM(V,1);
     BEGIN;
        DECLARE Z(N) FIXED BINARY,
           PRM ENTRY(FIXED BINARY,(*) FIXED BINARY) SETS(A,Z);
PRM:    PROCEDURE(K, Y) RECURSIVE;
           DECLARE (X(N), Y(N), K) FIXED BINARY,
              ROT ENTRY(FIXED BINARY, FIXED BINARY) SETS(Z);
ROT:       PROCEDURE(K, M) RECURSIVE;
              DECLARE (K, M) FIXED BINARY;
             IF M>N THEN RETURN;
             Z=X; Z(K)=X(M); Z(M)=X(K);
             CALL PRM(K+1, Z);
             CALL ROT(K,M+1);
           END ROT;
           X=Y;
           IF K=N THEN DO;
              A(I,*)=X; I=I+1;
           END;
           ELSE CALL ROT (K, K);
        END PRM;
        CALL PRM(1, V);
  END PERM;
```

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.

PB3.8 REFERENCES

PB3.8.1 Review: Gerald M. Weinberg, PL/I Programming Primer, McGraw Hill Book Company, New York, 1966, 278 pages, $5.95.

The general organization and flow of this text is quite good particularly in that the elementary ideas and concepts are used to motivate the discussion of the more complex items. The only apparent weakness of presentation is the approach of suggesting to the student the structure of the language and leaving him to assume the rest. This may lead the student to incorrect ideas of PL/I. Although this approach may be an efficient method of teaching, it demands the presence of an experienced and interested instructor. The carefully constructed exercises serve to clarify and solidify the concepts presented. The presence of these exercises mollify to some extent the misgivings this reviewer has about the precision of exposition.

R. C. Wick

PB3.8.2 Review: Eric A. Weiss, The PL/I Converter, McGraw Hill Book Company, 1966, 116 pages, Probable Price $3.95.

This textbook should prove valuable to the programmer who needs a working knowledge of PL/I in a short period of time. As the author states, however, a knowledge of FORTRAN is a necessary pre-requisite to the use of this text.

The presentation is quite clear and highlights the similarities between FORTRAN and PL/I. It also gives the reader a starting point for use of the full-power of PL/I.

There are some items missing which might have been valuable had they been included. Chief among the missing items are problems or exercises, a concise summary of FORTRAN versus PL/I notions, and the mention of efficiency considerations.

R. C. Wick

NOTICE: This material may be protected by copyright law (Title 17, U.S. Code). From the Charles Babbage Institute collections.