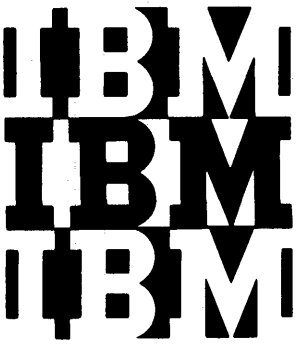


File No. S360-29
Form Y33-7006-0

Program Logic

IBM System/360 Conversion Aids: ALGOL-to-PL/I Language Conversion Program for IBM System/360 Operating System

Program Number 360C-CV-711

This document describes the internal logic of the ALGOL-to-PL/I Language Conversion Program for the IBM System/360 Operating System.

Program logic manuals are intended for use by IBM system and customer engineers involved in program maintenance, and by system programmers involved in altering the program design. Program logic information is not necessary for program operation and use. Therefore, distribution of this manual is limited to persons with program maintenance or modification responsibilities.

Restricted Distribution

PREFACE

This document describes the structure and functions of the ALGOL-to-PL/I Language Conversion Program: its components, their functions, and the control flow among them. The detailed organization of each component and the instructions used to implement its functions are given in the program listing. The object of this manual is to help the user to find any portion of the listing he requires.

In this manual, the IBM System/360 Operating System ALGOL language is referred to as ALGOL and the IBM System/360 Operating System PL/I (F) language is referred to as PL/I.

The manual consists of six sections. Section 1 is an introduction to the Language Conversion Program. Section 2 describes the Control Phase of the program. Sections 3, 4, 5, and 6 describe the phases into which the program is divided.

The reader must have a working knowledge of the ALGOL and PL/I languages and the System/360 Operating System.

The reader should have some familiarity with the contents of the following publications:

IBM System/360 Operating System, ALGOL Language, Form C28-6615[1]
IBM System/360 Operating System, PL/I (F) Programmer's Guide, Form C28-6591
ALGOL-to-PL/I Language Conversion Program for IBM System/360 Operating System, Form C33-2000[2]

In addition, the reader will find it useful to be acquainted with the publications:

IBM System/360 Operating System, Job Control Language, Form C28-6539.
IBM System/360 Operating System, PL/I Language Specifications, Form Y33-6003

---

[1]Referred to in the text as "the ALGOL language manual"

[2]Referred to in the text as "the ALGOL-to-PL/I language conversion manual"

First Edition (January, 1969)

Specifications contained herein are subject to change from time to time. Any such changes will be reported in subsequent revisions or Technical Newletters.

Request for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM France, Centre d'Etudes et Recherches, Programming Publications, Department 843, 06 - La Gaude, France.

The ALGOL-to-PL/I Language Conversion Program, henceforth referred to in this publication as "the LCP", helps the user to transfer to PL/I by converting System/360 Operating System ALGOL source programs into PL/I (F) programs. In this publication, the converted PL/I programs are referred to as "target" programs. The LCP itself is written in PL/I (F) language. It is distributed in object module form for inclusion in the user's library.

The Job Control Cards used to call the program from the library are described in the ALGOL-to-PL/I language conversion manual.

The ALGOL source program must be free from errors; i.e., it must meet the following requirements:

- It has been successfully compiled, without producing error messages, into a System/360 machine code version that produces the results expected by the user.
- It consists of statements that are in strict conformity with the syntactic and semantic rules laid down in the ALGOL language manual.

Since these requirements are assumed, the LCP makes no attempt to detect errors, nor to continue scanning after an error has appeared, nor to give precise diagnostics. The LCP may, however, attempt to process an ALGOL program containing an error (where, for example, a card has been modified after compilation). In this case, conversion may stop altogether or, if carried out, may result in a PL/I program of dubious meaning and validity. In addition, note that:

- If the ALGOL program contains function designators that produce side-effects, the converted program may produce different and unpredictable side-effects.
- If, for any reason, a source statement cannot be converted (see Appendix D of the language conversion manual), a message to that effect is printed in the output listing.

The LCP is divided into the following six phases:

1. Control Phase
2. Phase 10, which performs the following functions:
   - Converts the "hardware representations" used in ALGOL into internal form
   - Builds a dictionary of identifiers
   - Detects the block structure of the program
   - Converts the elements of the program that do not require reference to the dictionary, e.g., comments, strings, and certain declarations.
3. Phase 20, which searches in the dictionary created during Phase 10, for the types of the identifiers
4. Phase 25 (optional), which completes the dictionary if necessary
5. Phase 30 (optional), which generates procedure declarations, using the dictionary created by Phase 10 and, if necessary, completed by Phase 25
6. Phase 40, which generates the PL/I statements, and all comments. Phase 40 converts all the elements not converted by Phases 10 and 30, statement by statement, using, when necessary, the dictionary of identifiers built by Phase 10, and the identifier descriptions added by Phase 20.

TABLES

The LCP uses a number of tables. The way in which these tables are used by the individual routines is described in the relevant sections. Appendix A contains a full description of the most important tables.

OVERALL LOGIC OF THE LCP

Figure 1 illustrates the overall internal logic of the LCP and shows the relationship among the phases of the program.

The Initialization Procedure of the Control Phase receives control from the System/360 Operating System and then calls Phase 10.

Phase 10 takes the ALGOL source program from SYSIN as its input. It scans the source program, placing the intermediate text on SYSUT1, and builds the dictionary, which is placed on SYSUT2. Optionally, Phase 10 can list the source program on SYSPRNT. It finally gives control back to the Control Phase, which calls Phase 20.

Phase 20 takes the intermediate text and the dictionary as its input. It inserts the identifier types in the intermediate text, and, in certain cases, builds a table which is placed on SYSUT1. Control again returns to the Control Phase.

The Control Phase now tests for the presence of procedure declarations in the source program. If the result of this test is positive, and if, in addition, certain conditions are present, the Control Phase calls Phase 25 to complete the dictionary, using the table created by Phase 20. (For details of the requisite conditions, see the section describing Phase 25.) Control returns to the Control Phase.

The Control Phase then calls Phase 30 which generates the PL/I PROCEDURE statements. Note that:

1. Phase 30 may be called directly after Phase 20 if Phase 25 is not required.
2. If the source program contains no procedure declarations, neither Phase 25 nor Phase 30 are required.

Phase 40, the last phase of the program, is therefore called, via the Control Phase, either after Phase 30 or directly after Phase 20. Phase 40 generates the PL/I statements and comments, taking the modified intermediate text on SYSUT1 as its input. On completion of Phase 40, control returns for the last time to the Control Phase which uses the Return Procedure to generate error messages if necessary. (An error message may have been issued during a previous phase if a syntactic error was detected in the program.)

Control finally returns to the Operating System.

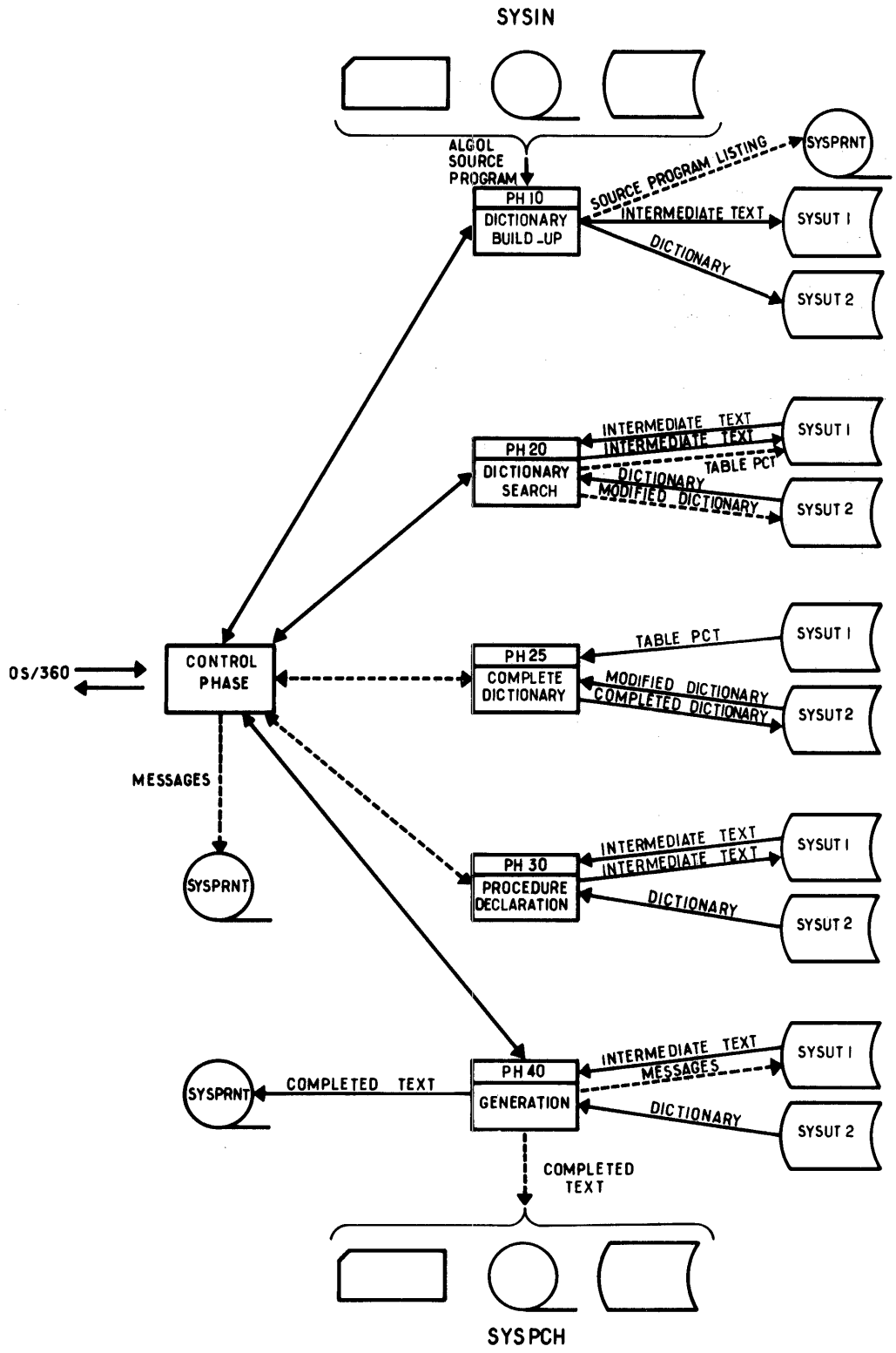


Figure 1. Overall Logic of the LCP

CONTROL PHASE

The LCP Control Phase (see Chart 001) receives control from the System/360 Operating System by means of an EXEC control card. This phase controls the flow of the program, using two procedures:

1. Initialization Procedure
2. Return Procedure

The actions performed by these procedures are described below.

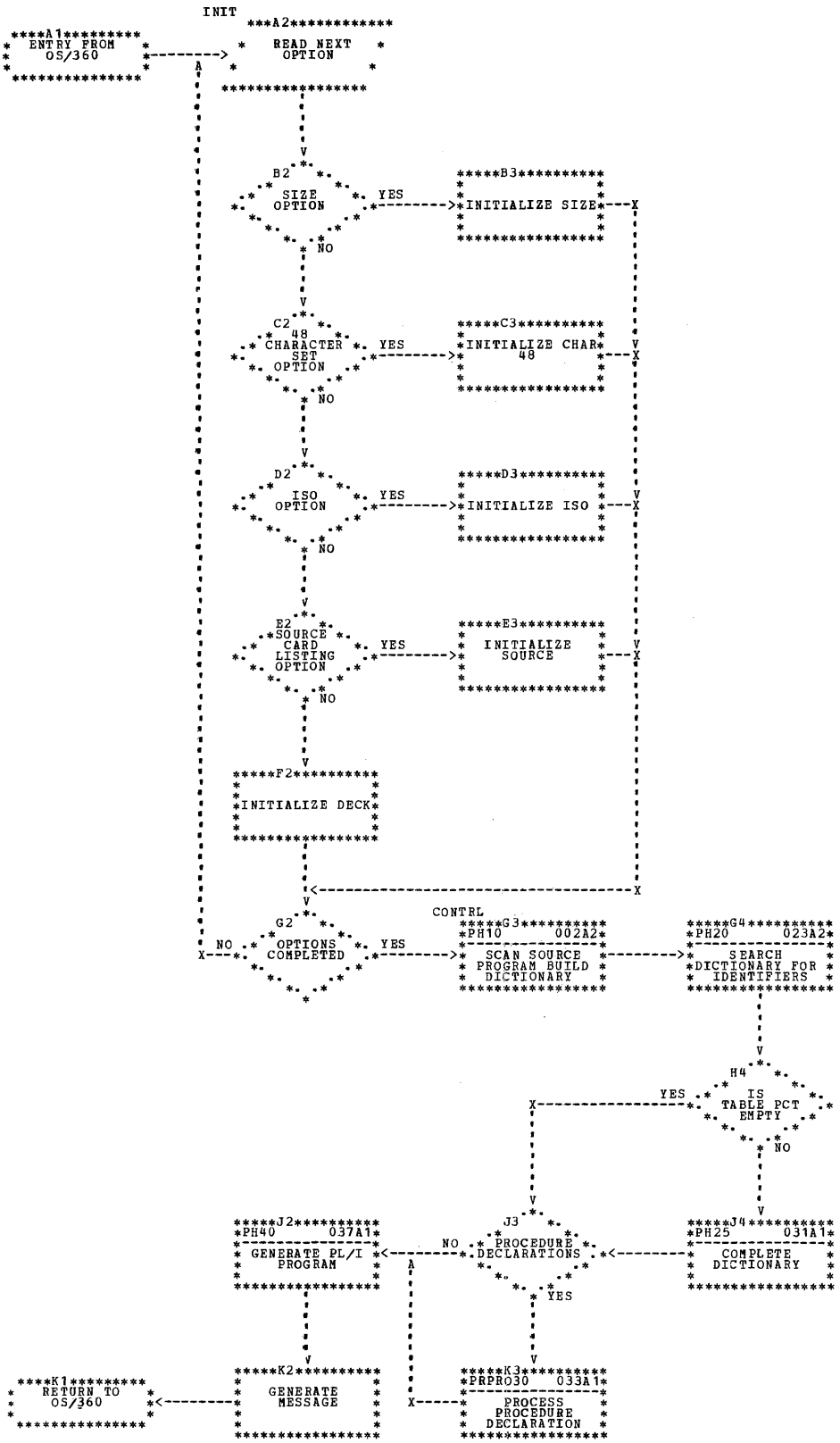
Initialization Procedure

The Initialization Procedure initializes the settings of the switches in the LCP communication area. These are set either to a standard status, or to the status indicated by the user in his EXEC control invokes Phase 10.

RETURN PROCEDURE

The Return Procedure, which is entered on completion of each phase of the program, calls the following phase. On completion of Phase 40, the Return Procedure generates error messages where necessary and restores control to the System/360 Operating System. Should the program stop at any time during execution because an error has been detected, the Return Procedure automatically regains control and issues an appropriate message. Control then returns to the Operating System.

Chart 001. Overall Logic of Control Phase



PHASE 10

Phase 10 performs three main functions:

1. It acts as a preprocessor by converting into internal form the ALGOL "hardware representations" (see Chapter 1 and Appendix 2 of the ALGOL language manual) and by converting into PL/I certain basic elements of the source program.

2. It detects the block structure of the program.

3. It builds a dictionary of declared or specified identifiers. This dictionary serves for the conversion of assignment statements, subscript expressions (other than constants), and actual parameters (called by value).

The overall logic of this phase is illustrated in Chart 002.

Preprocessing

The input to Phase 10 is the ALGOL source program on SYSIN. After initializing its internal parameters and tables, Phase 10 scans the ALGOL program to perform the following actions:

- ALGOL basic symbols with a hardware representation that contains more than one character are recognized and converted into fixed-length internal form.
- Blanks outside strings and comments are eliminated.
- Strings and various forms of comments are converted into their PL/I equivalents. When the sequence */ appears within the text of a comment, it is converted into *-. In an open string, a single quotation mark is replaced by double quotation marks.
- Identifiers over six characters in length are truncated to six characters. (Identifiers in the intermediate text always contain six characters; blanks are added if necessary.)
- Certain delimiters (e.g. 'END', semicolon, and 'DO') are recognized and used for delimiting essential components of the ALGOL program, such as declarations and statements. An internal reference number is added to each of these elements.

Detection of Block Structure

The nested program structure is detected by placing the appropriate delimiters in a stack, and by setting an indicator in the intermediate text. The block structure of the source program defines the scope of the identifiers. This procedure is used also to recognize the end of 'FOR' statements and of procedure declarations.

The LCP handles a procedure declaration seperately from the rest of the block in which it occurs, and considers it as a new block. A number is assigned to each block and to each procedure declaration. The dictionary (see Appendix A), is thus divided into two sections:

1. DICEXT, which contains entries relative to blocks.

2. DICPRO, which contains entries relative to procedure declarations.

At the end of a program-block, the corresponding section is placed on SYSUT2, using a key to identify the section. (See Appendixes A and B for further details.)

Construction of Dictionary

Phase 10 builds the dictionary using the source program declarations and specifications.

Identifiers are checked, before being entered in the dictionary, to prevent conflict with built-in PL/I function-names or with relational operators when the 48-character set is being used. In the event of a possible conflict, an appropriate flag is entered in the dictionary.

In the case of a procedure declaration, the description of the formal parameters is completed using the specifications. The declaration processing routine attempts to convert as much as possible of the declaration. If a switch-declaration cannot be converted, a flag to that effect is placed in the intermediate text.

The output from Phase 10 is an intermediate text, placed on SYSUT1, using the internal reference number as a key.

PHASE 10 ROUTINES

To perform its various functions, Phase 10 uses a number of routines. These fall into the following two categories:

1. Input/output routines
2. Processing routines

INPUT/OUTPUT ROUTINES

These are used by Phase 10 to perform various simple and frequently recurring functions, as described below:

BUFPUT -- places the contents of the output buffer on SYSUT1.

DICPUT -- places dictionary DICEXT on SYSUT2.

GENER -- places in the output buffer characters that have been stored in the intermediate area.

GETCAR -- places in the word TCHAR the character in the input buffer, positions a pointer to the next character, and if necessary, reads the next card.

GQUOT -- reads the ALGOL keyword enclosed by quotes, converts it into internal code, and enters the converted keyword in CODCAR. Determines the type of the keyword (declarator, specificator, etc.) and enters the information obtained in LUC.

PRINT -- lists the ALGOL source program on SYSPRNT.

PROPUT -- places dictionary DICPRO on SYSUT2.

PUTCAR -- places in the output buffer the character contained in CODCAR and positions a pointer to the next character.

FINPG -- End-of-phase routine, which closes the files, places the contents of the last output buffer on SYSUT1, and returns control to the Control Phase.

PROCESSING ROUTINES

This section contains detailed descriptions of the routines used by Phase 10 to process various elements of the ALGOL source program. The information given about each routine is set out as follows:

- Purpose of the routine
- Calling phase or routine(s)
- Processing
- Routine(s) called
- Exit from the routine

In order to simplify reference, the routines are described in alphabetical order. Each routine is illustrated in a separate flowchart.

ARRAYS ---------- Charts 003,004

Purpose: To process the text following the ALGOL declarator 'ARRAY'.

Called by: DECLAR.

Processing: Generates the list of array-names, converts the bound-pair-list into internal code and, depending on the type of the list, generates one of the following:

- FLOAT(16)
- FIXED BINARY(31)
- BIT(1)

Enters in DICEXT the information concerning each identifier of the array-name list.

Routines called: BUFPUT, GLABEL, GENER, GETCAR, PUTCAR.

Exit: DECLAR.

COMENT ---------- Chart 005

Purpose: To process comments.

Called by: QUOT.

Processing: Places any text between 'COMMENT' and a semicolon into BUFFER, enclosing this text between the delimiters /* and */. Replaces the sequence */, when it occurs in the ALGOL program, by *-.

Routines called: BUFPUT, GENER, GETCAR, PVIRGU.

Exit: SCAN.

DECLAR_____Charts 006,007

Purpose: To process ALGOL declarators.

Called by: QUOT.

Processing: Detects the type of the declarator(s), places in area TYPCAL the information pertaining to the declaration. If one of the declarators is 'ARRAY', 'PROCEDURE', or 'SWITCH', goes to the routine ARRAYS, PROCED, or SWITCH respectively. If not, enters identifiers of a same type, together with the type, in dictionary DICEXT. Depending on the declarator, generates one of the following:

- DECLARE ... FLOAT
- FIXED BINARY(31)
- BIT(1)

If necessary, updates the stack.

Routines called: DICPUT, GENER, GETCAR, GLABEL, GQUOT.

Exit: ARRAYS, PROCED, SCAN, SPE1, SWITCH.

DEPOIN_____Chart 008

Purpose: To process the following characters: : := =

Called by: SCAN.

Processing: Detects whether the character sequence to be processed is a colon , an equal sign, or a combination of the two. Converts the character into internal code and places it into BUFFER.

Routines called: GETCAR, PUTCAR, SCAN.

Exit: SCAN or SCA1.

DESPEC_____Chart 009

Purpose: To process a sequence of two special characters.

Called by: SCAN.

Processing: If the first character is a semicolon, calls routine PVIRGU to process it. If the first character is a period, goes to routine UNPOIN. If it is neither of these:

a. If the combination of the two characters constitute an ALGOL symbol, converts the symbol into internal code and places it into BUFFER.

b. If the two characters do not constitute an ALGOL symbol, converts the first character only and places it into BUFFER.

Routines called: PUTCAR, PVIRGU, SCAN, UNPOIN.

Exit: SCAN or SCA2.

GLABEL_____Chart 010

Purpose: To place the current identifier in areas IDENT1 and IDEXT6.

Called by: ARRAYS, DECLAR, LETR, PROCED, SPECIF, SWITCH.

Processing: Ignoring blanks, fetches the characters making up the identifier, and truncates the latter to six characters. Tests whether or not a $ sign should be added as suffix to the identifier, and adds $ if required. Calculates length of the identifier.

Routine called: None.

Exit: Calling routine.

LEND_____Chart 011

Purpose: To process the ALGOL word 'END' and the comments, which, in certain cases, follow it.

Called by: QUOT.

Processing: Tests the contents of the stack. If it finds that the last symbol in the stack indicates the beginning of a loop, it generates END. If it finds that the last symbol indicates a block, it places DICEXT on SYSUT2 and generates END. Tests for the beginning of a comment; if a comment is found, places it into BUFFER, enclosing it between the delimiters /* and */. If the comment is followed by:

- 'ELSE', generates a ;ELSE sequence, and returns to SCAN.
- A semicolon, generates a semicolon, processes it using routine PVIRGU, and returns to SCAN.

- 'END', generates a semicolon and returns to LEN1.

Routines called: BUFPUT, DICPUT, GENER, GETCAR, PVIRGU, QUOT.

Exit: LEN1, SCAN.

LETR Chart 012

Purpose: To process identifiers.

Called by: SCAN.

Processing: Reads the identifier, using routine GLABEL, and places it into BUFFER.

Routines called: BUFPUT, GLABEL, SCAN.

Exit: SCA2.

PAREN Chart 013

Purpose: To generate letter-strings that occur in procedure or function declarations or in procedure or function calls.

Called by: PROCED, SCAN.

Processing: Generates the letter-string, enclosing it between the delimiters /* and */.

Routines called: GENER, GETCAR.

Exit: Calling routine.

PROCED Chart 014

Purpose: To process procedure declarations.

Called by: DECLAR.

Processing: Detects whether or not the procedure contains parameters, and updates the stack accordingly. If parameters are found, processes them. Enters information relating (a) to procedure identifiers in dictionary DICEXT, (b) to parameter identifiers in dictionary DICPRO. Generates procedure declarations.

Routines called: DECLAR, GENER, GETCAR, GLABEL, PAREN.

Exit: DECLAR.

PVIRGU Chart 015

Purpose: To generate an end-of-loop, an end-of-procedure, or an end-of-function, depending on the contents of the stack.

Called by: DESPEC, LEND, UNPOIN, VNSPEC.

Processing: Tests the last item in the stack. Then, if the last item in the stack is:

- DO, generates an END; sequence. Goes on to test the preceding item, and so forth.
- A non-type procedure containing parameters, places DICPRO on SYSUT2, and generates: END;
- A type procedure containing parameters, places DICPRO on SYSUT2, and generates: RETURN($RTURN);END;
- A non-type procedure with no parameters, generates: END;
- A type procedure with no parameters, generates: RETURN($PTURN);END;

If the program is an ALGOL procedure, enters the name and type of the procedure in PGPROL and PGPROT respectively for transmission to Phase 20.

Routine called: GENER.

Exit: Calling routine.

QUOT Chart 016

Purpose: To process the following characters or groups of characters: quote separator (exponent part), '/', '(' followed by an ALGOL string, ALGOL keyword enclosed by quotes.

Called by: LEND, SCAN.

Processing: Recognizes groups of characters beginning with a quotation mark.

- If it finds the quote separator (exponent part), converts it into internal form, and places it on SYSUT1.
- If it finds '/', converts it into internal form, and places it on SYSUT1.
- If it finds '(', places the string that follows, up to the last ')', into BUFFER, between quotes.

- If it recognizes the group of characters as an ALGOL keyword, and if:

  a. It finds specifiers or the word 'CODE', goes to routine SPECIF.

  b. It finds any declarator except 'CODE', goes to routine DECLAR.

  c. It finds 'COMMENT', goes to routine COMENT.

  d. It finds 'END', goes to routine LEND.

  e. It finds 'DO', 'THEN', or 'ELSE', processes them.

  f. It finds any other ALGOL keyword, converts it into internal form and places it into BUFFER.

Routines called: BUFPUT, COMENT, DECLAR, GENER, GETCAR, GQUOT, LEND, PROPUT, PUTCAR, SCAN, SPECIF.

Exit: COMENT, DECLAR, LEND, SCAN, SCA1, SPECIF.

SCAN ---------- Chart 017

Purpose: To call the appropriate routine required to process a character.

Called by: All routines except ARRAYS, FINPG, PROCED, SWITCH.

Processing: Reads and tests the character.

- If it is a letter, goes to routine LETR.
- If it is a digit or a left parenthesis, processes it.
- If it is a colon or an = sign, goes to routine DEPOIN.
- If it is a quotation mark, goes to routine QUOT.
- If it is any other special character, tests whether it is the first of a group of two special characters; if so, goes to routine DESPEC, if not, goes to routine UNSPEC.

Routines called: DEPOIN, DESPEC, GENER, GETCAR, PAREN, PUTCAR, QUOT, VNSPEC.

Exit: DEPOIN, DESPEC, LTR, QUOT, SCAN, SCA2, VNSPEC.

SPECIF ---------- Charts 018,019

Purpose: To process the following:

- 'CODE'
- Text following specifier
- Value part

Called by: DECLAR, QUOT.

Processing: There are two entry-points to this routine: SPECIF and SPE1. At entry-point SPECIF, processes a specificator or the word 'CODE'. Generates the appropriate PL/I declaration and enters the relevant information in DICPRO.

At entry-point SPE1, processes the keywords used as specifiers which are also used as declarators. Partially computes TYPCAL and enters the information obtained in DICPRO and in DICEXT, which already contains the procedure type.

Routines called: GENER, GETCAR, GLABEL, GQUOT, PROPUT, SCAN.

Exit: SCAN.

SWITCH ---------- Chart 020

Purpose: To process the text following the declarator 'SWITCH'.

Called by: DECLAR.

Processing: Ascertains which identifiers follow the declarator 'SWITCH', and generates the declaration LABEL INITIAL. If necessary, places number of error message in RMES.

Routines called: GLABEL, GENER, GETCAR, SCAN.

Exit: SCAN.

UNPOIN ---------- Chart 021

Purpose: To process groups of special characters beginning with a period.

Called by: DESPEC.

Processing: Ascertains whether or not the period is the first of a group of two characters representing an ALGOL symbol.

- If not, i.e., if the period appears on its own, converts it into internal code and places it into BUFFER.

- If so, and if the period is followed by another period (..), converts the group into internal code and places it on SYSUT1.

- If so, and if the period is followed by a comma (.,), calls the routine PVIRGU to process the group

Routines called: GETCAR, PUTCAR, PVIRGU, SCAN.

Exit: SCAN, SCA1, SCA2.

VNSPEC Chart 022

Purpose: To process isolated special characters.

Called by: SCAN.

Processing: If the character is a semicolon, calls routine PVIRGU to process it. In all other cases, converts the character into internal code and places it into BUFFER.

Routines called: PUTCAR, PVIRGU, SCAN.

Exit: SCA6.

Chart 002. Overall logic of Phase 10

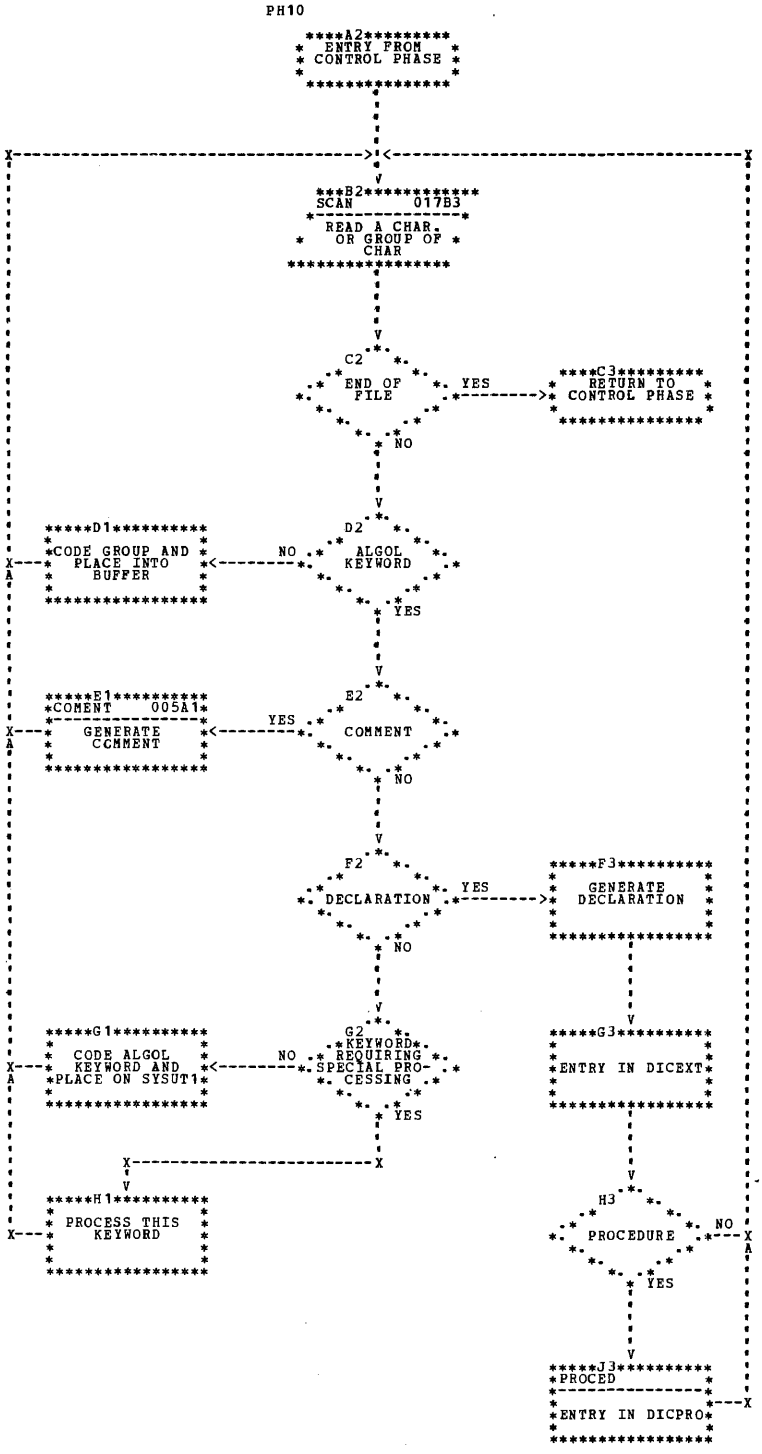


Chart 003. ARRAYS Routine, Part 1 of 2

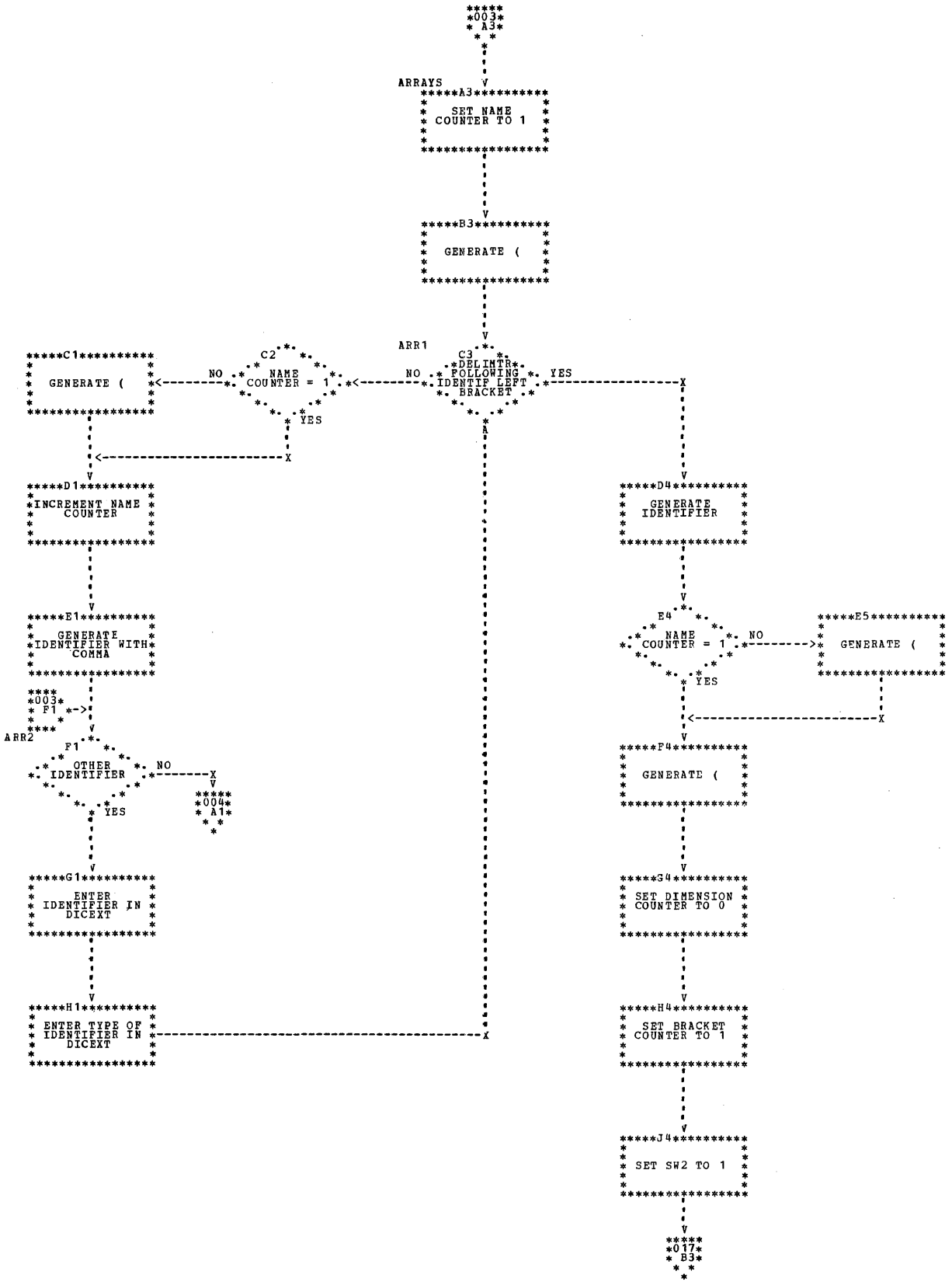


Chart 004. ARRAYS Routine, Part 2 of 2

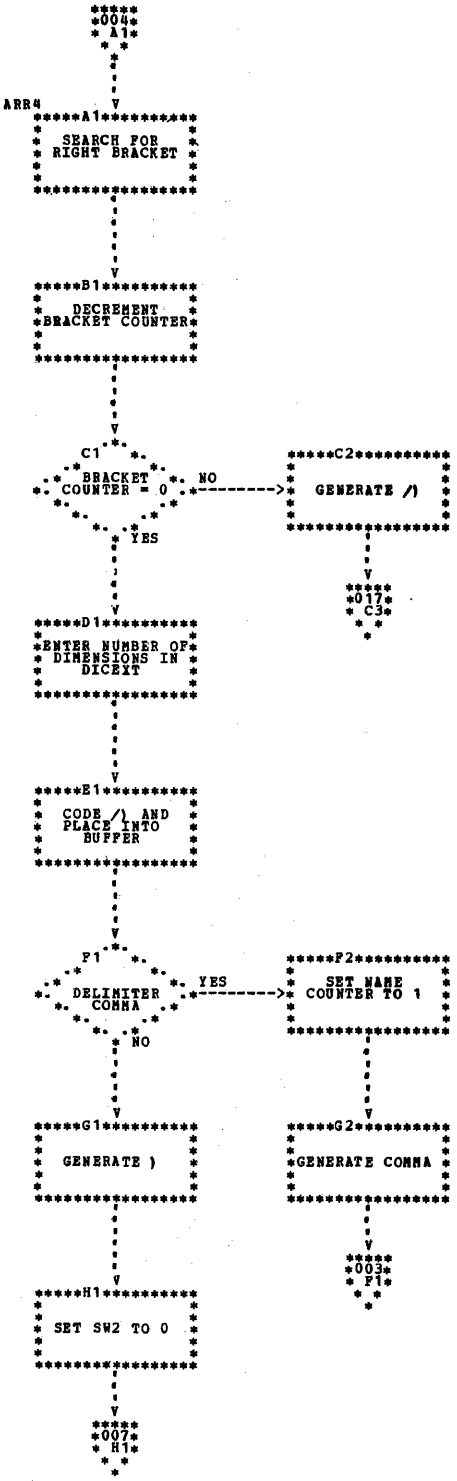


Chart 005. COMENT Routine

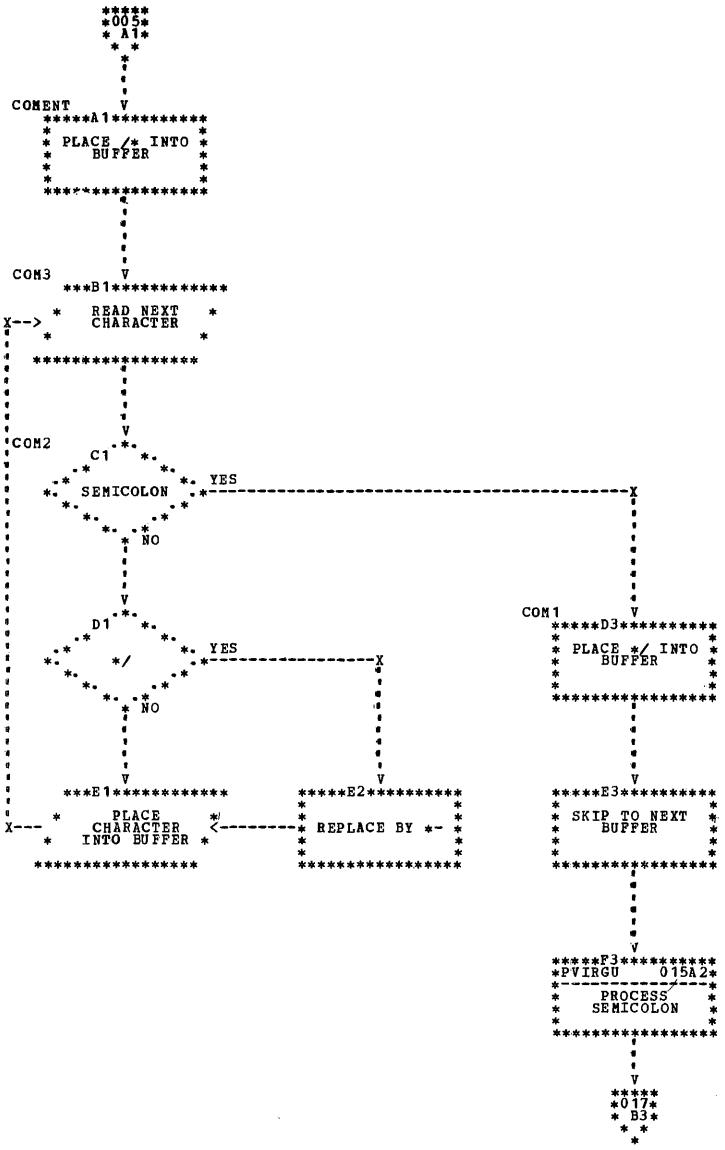


Chart 006. DECLAR Routine, Part 1 of 2

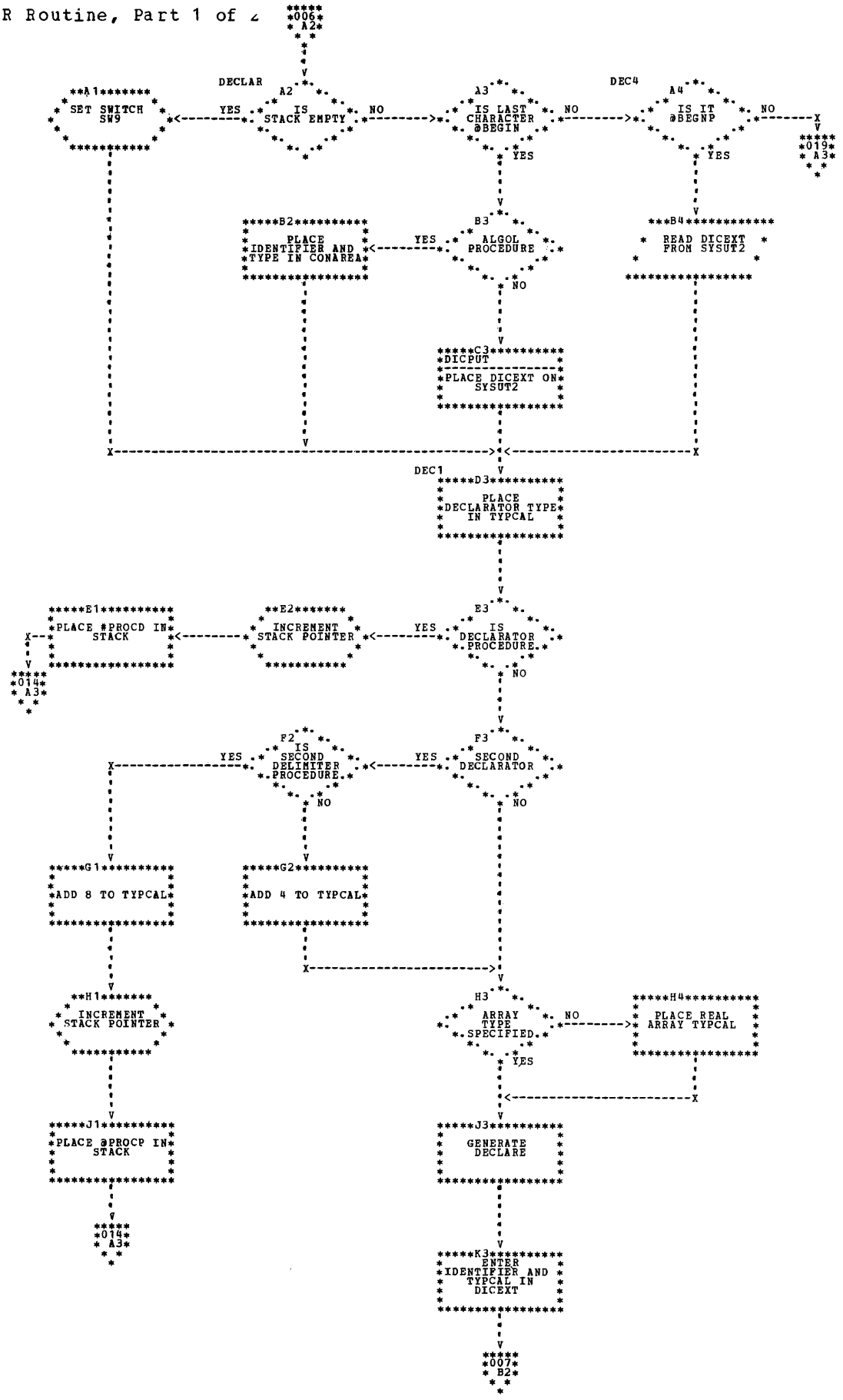


Chart 007. DECLAR Routine, Part 2 of 2

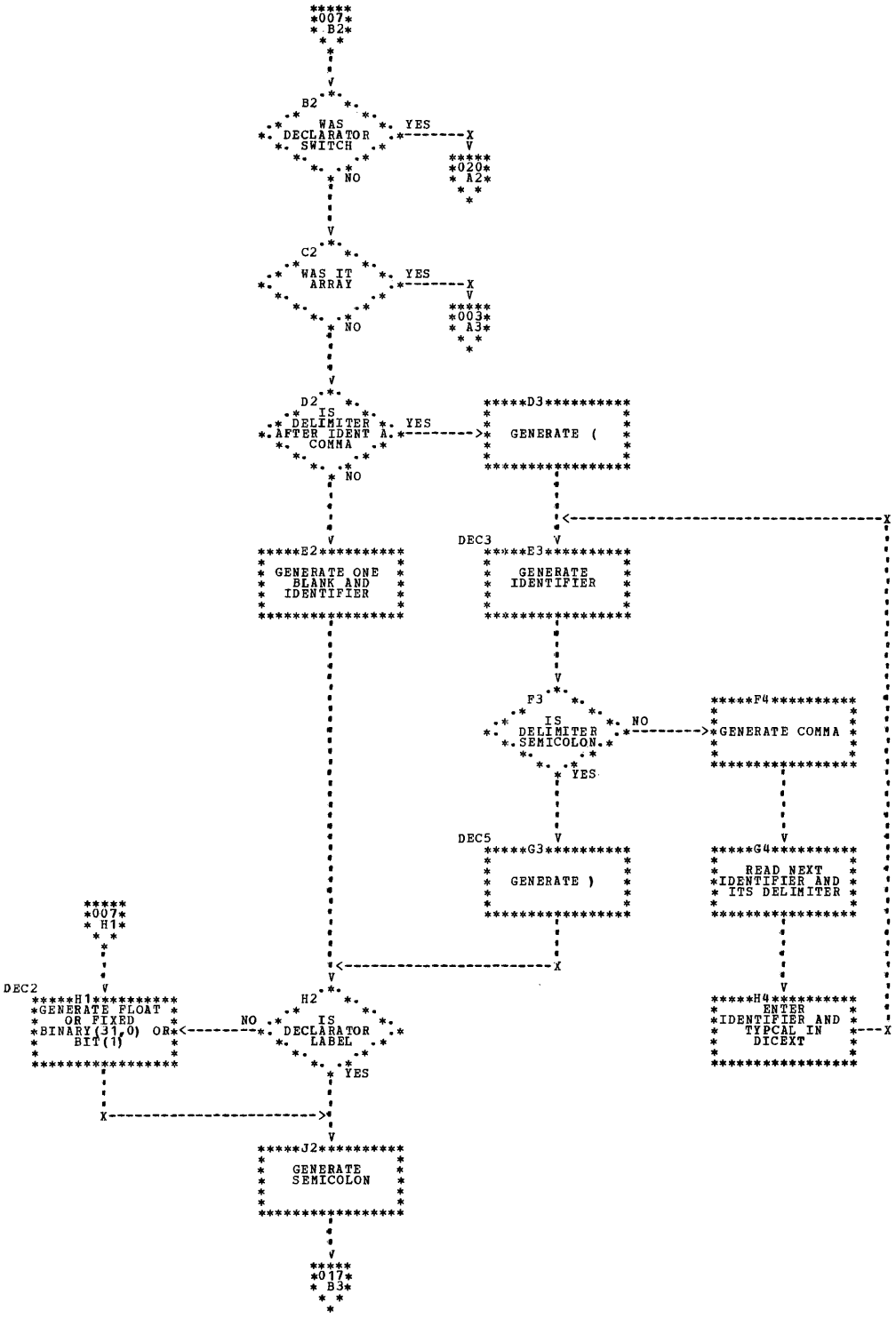


Chart 008. DEPOIN Routine

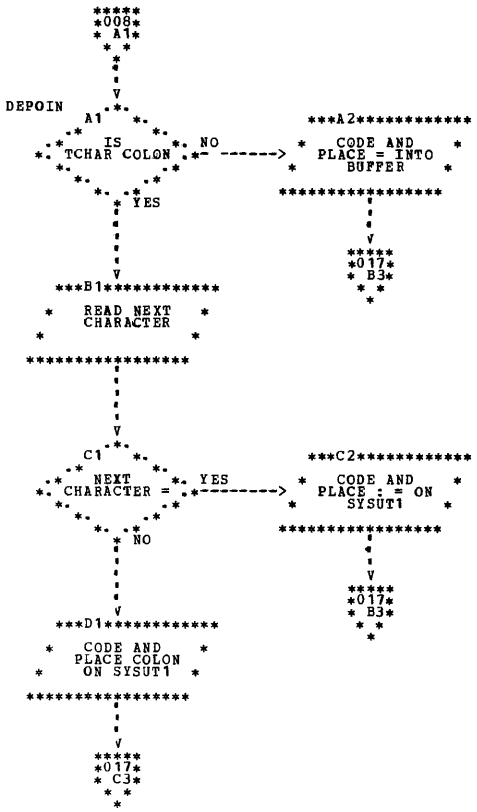


Chart 009. DESPEC Routine

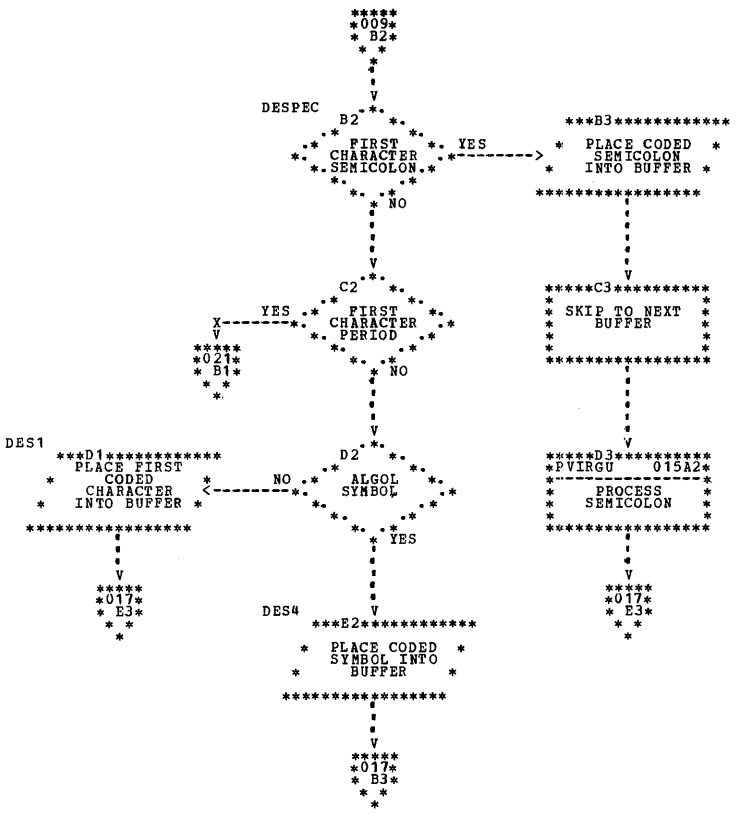


Chart 010. GLABEL Routine

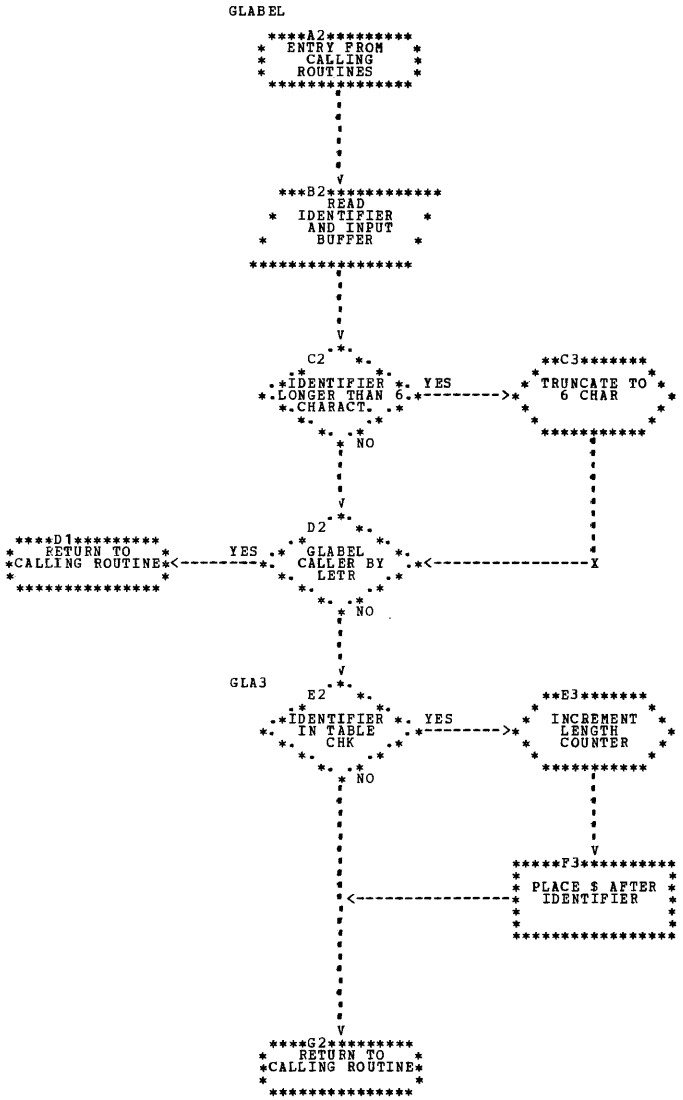


Chart 011. LEND Routine

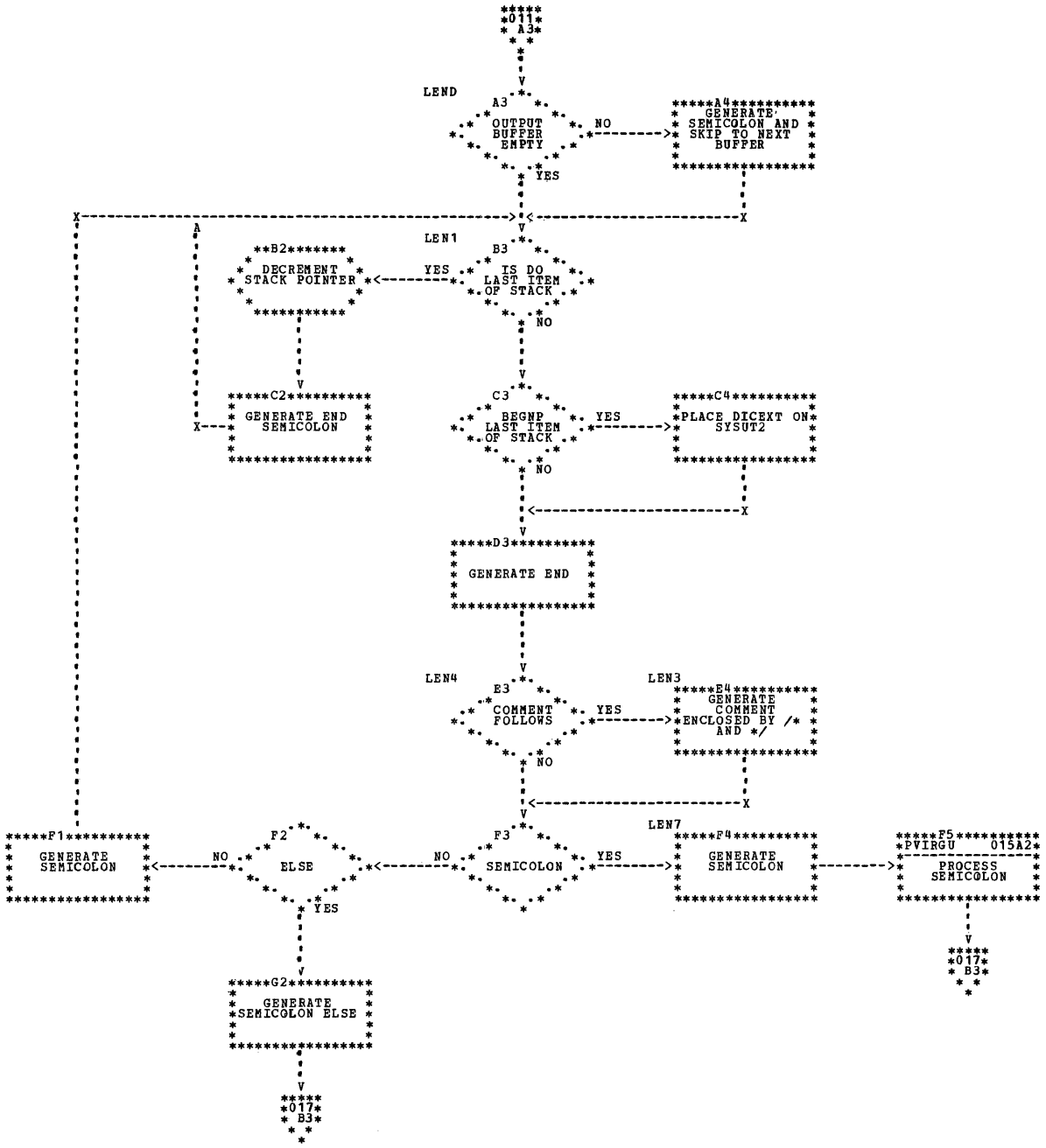


Chart 012. LETR Routine

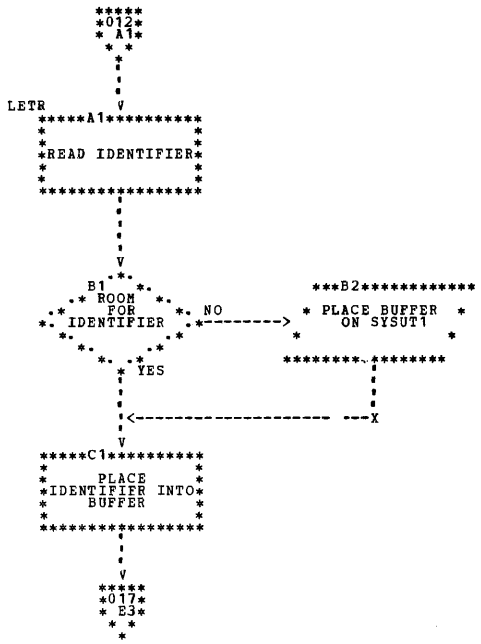


Chart 013. PAREN Routine

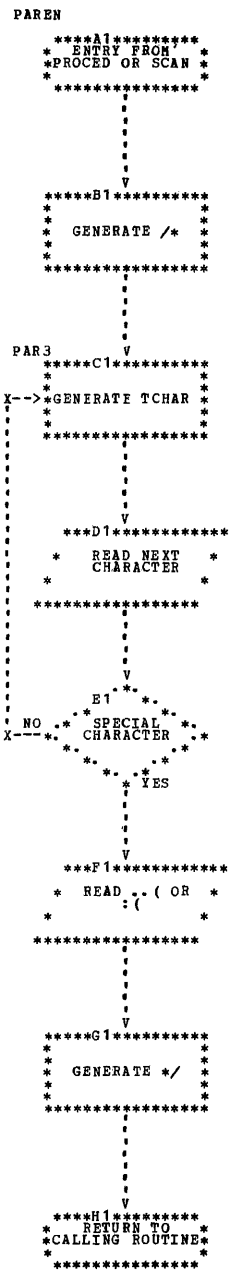


Chart 014. PROCED Routine

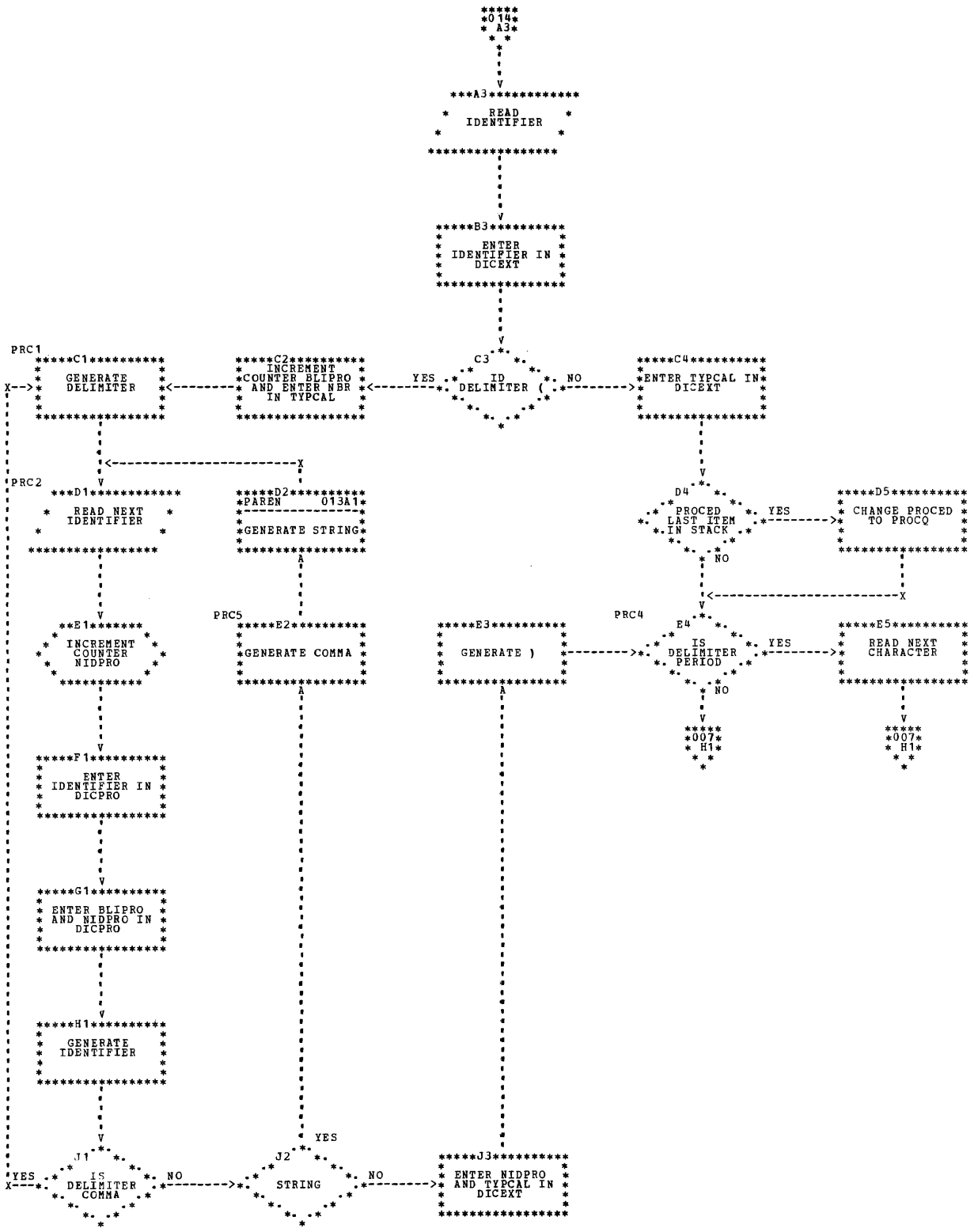


Chart 015. PVIRGU Routine

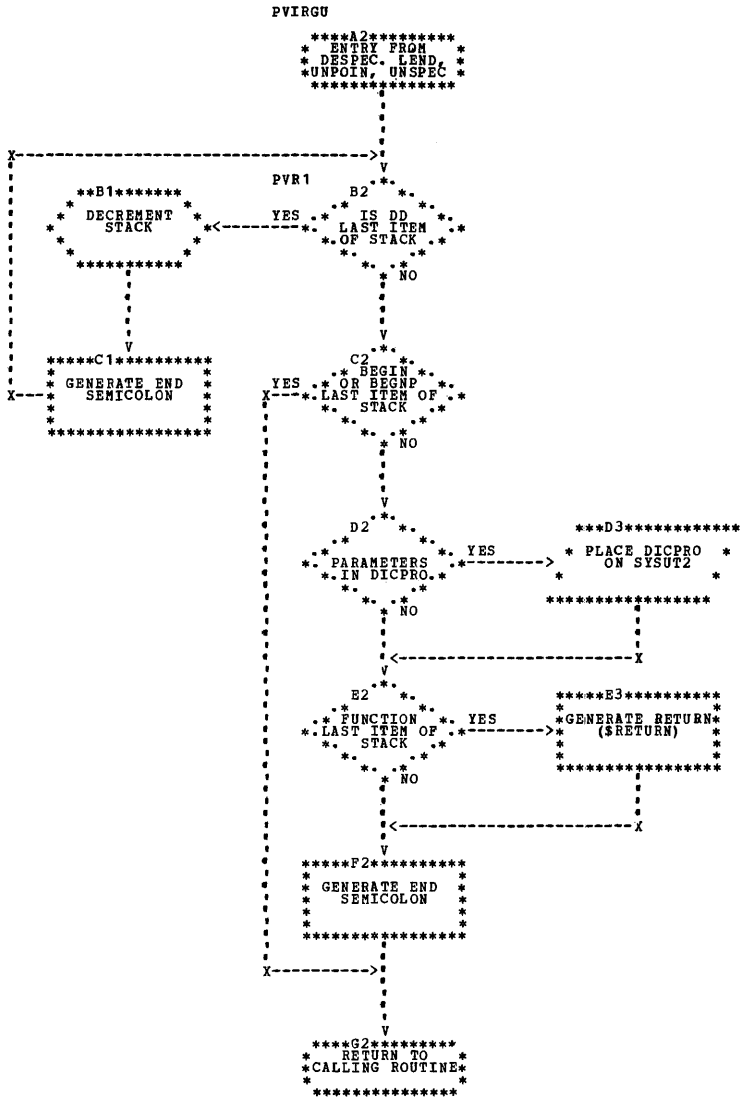


Chart 016. QUOT Routine

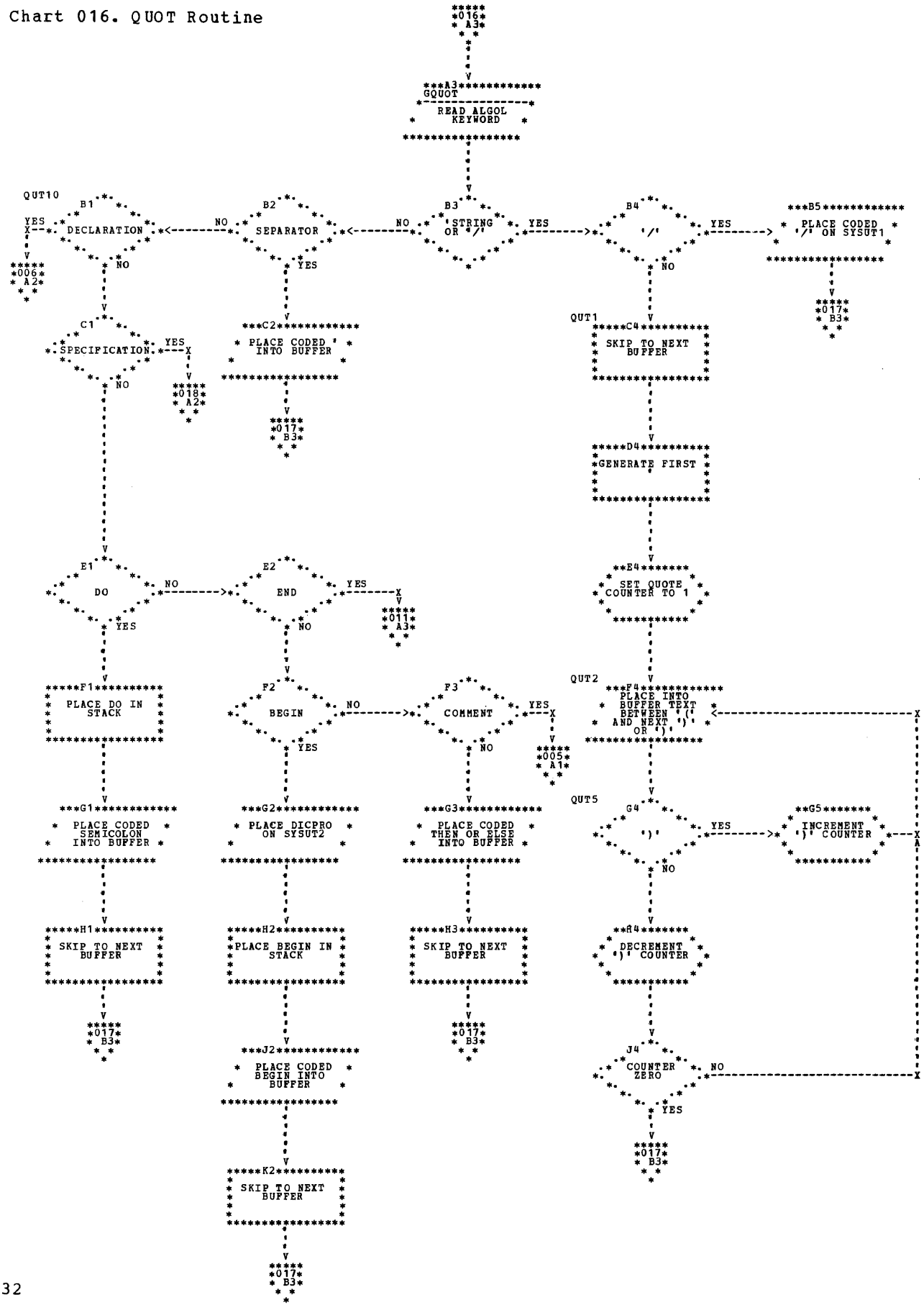


Chart 017. SCAN Routine

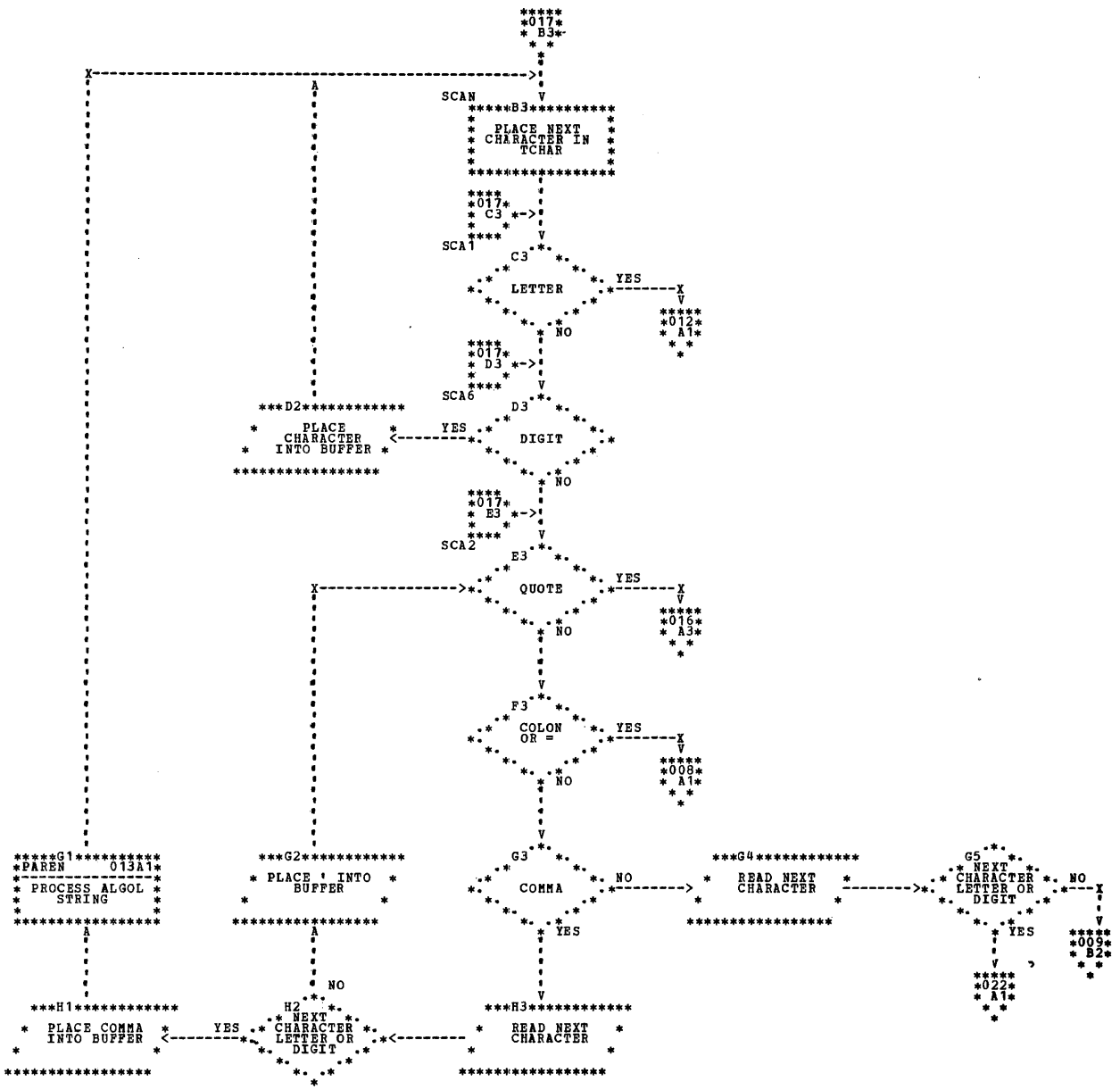


Chart 018. SPECIF Routine, Part 1 of 2

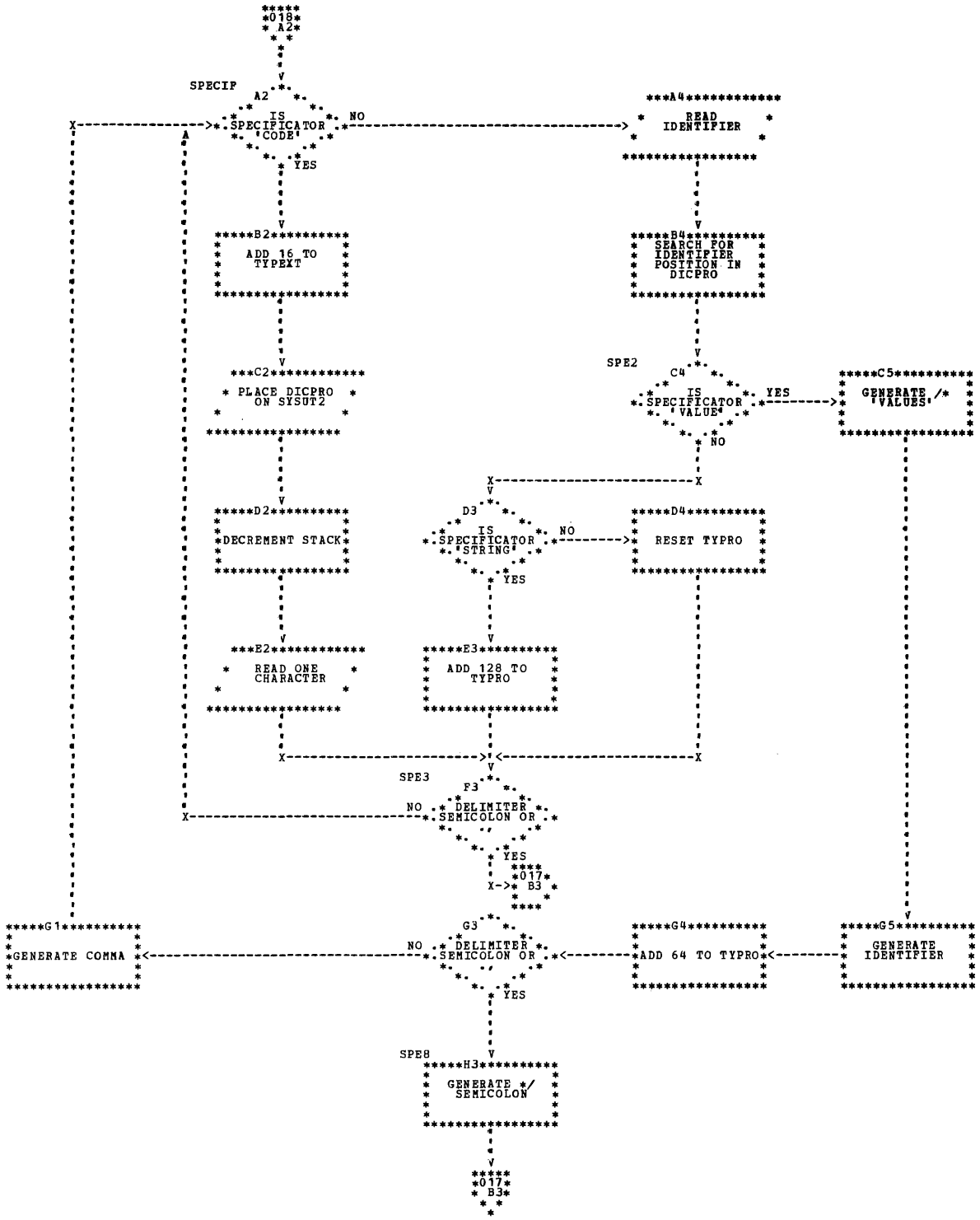


Chart 019. SPECIF Routine, Part 2 of 2

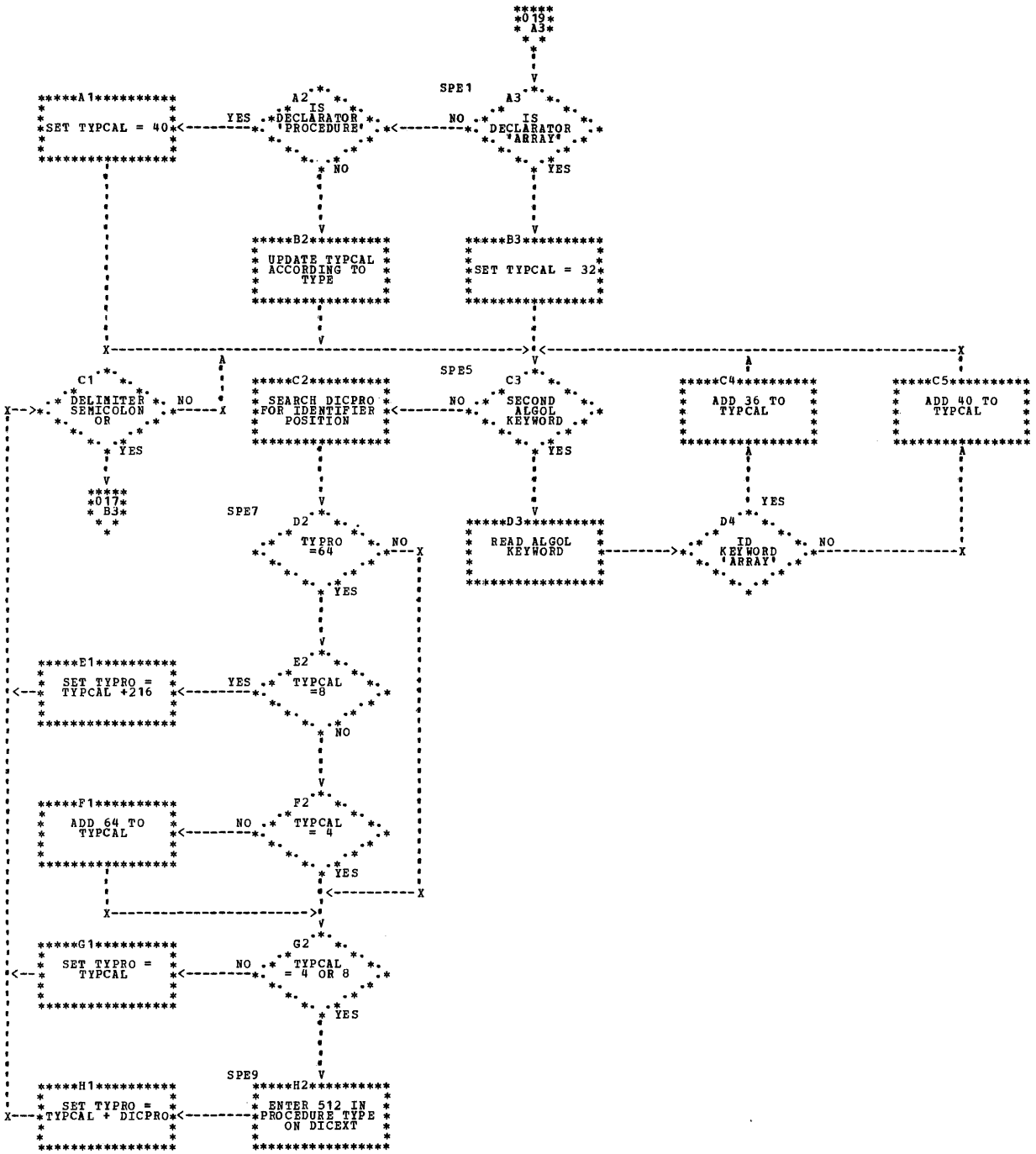


Chart 020. SWITCH Routine

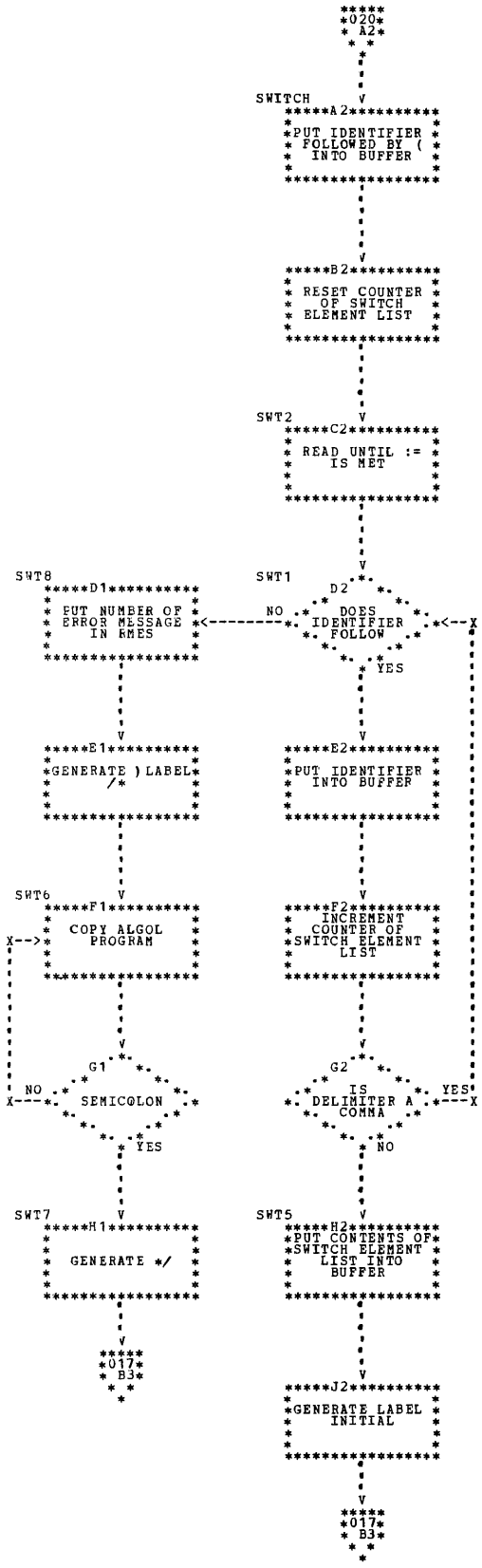


Chart 021. UNPOIN Routine

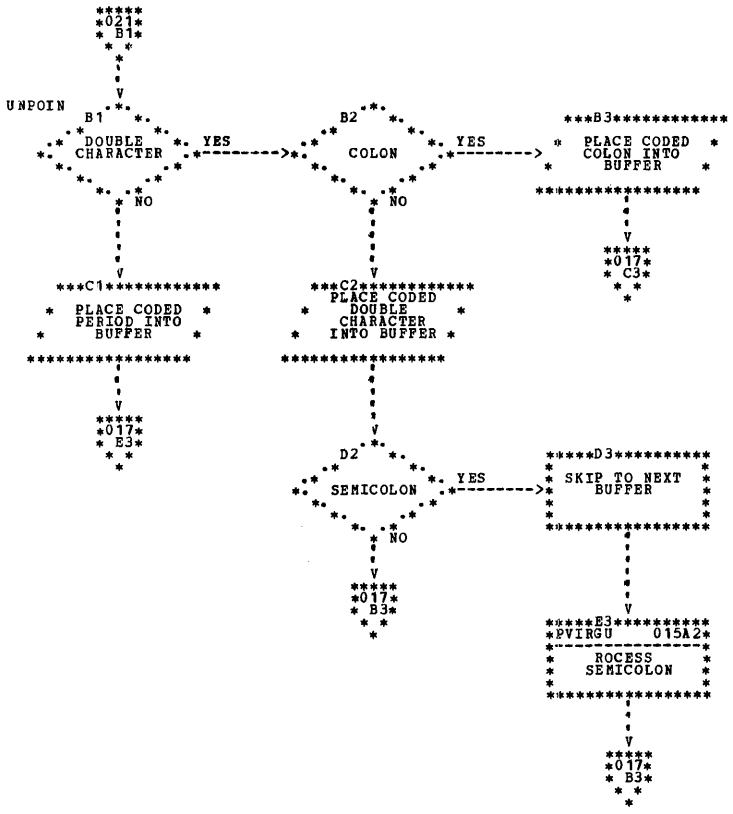
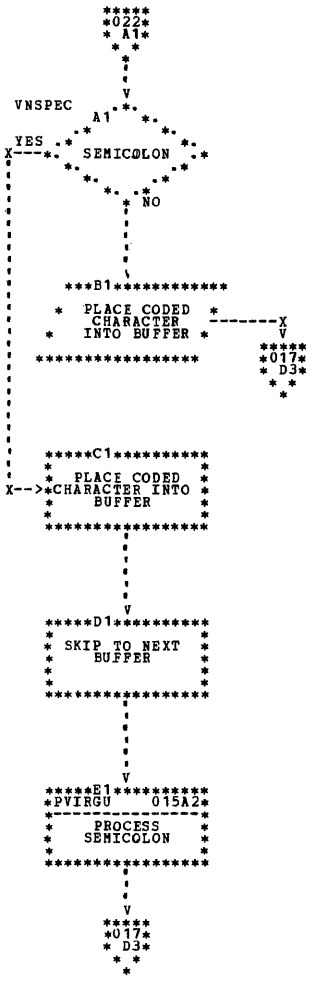


Chart 022. VNSPEC Routine



Phase 20 searches in the dictionary created during Phase 10 to discover the type of the identifiers. This information is placed in the intermediate text, each identifier being followed by its description. The overall logic of this phase is illustrated in Chart 023.

The input to Phase 20 consists of:

- The dictionary, on SYSUT2
- The intermediate text produced by Phase 10, on SYSUT1

Phase 20 cannot transfer its input directly from SYSUT2 to the dictionary because of temporary limitations in the PL/I compiler. DICEXT and DICPRO are buffers between the dictionary (DICT) in main storage and SYSUT2. The parts of the dictionary corresponding to DICEXT and DICPRO are also known as DICEXT and DICPRO in this section. For further details, see Appendix A.

When an identifier is not found in the dictionary and is followed by a left parenthesis, the search continues by checking for the presence of standard function and input/output procedure names.

A procedure identifier which occurs as left-part variable is replaced by the identifier $RTURN.

When a formal parameter specified by 'ARRAY' appears in an expression, Phase 20 calculates the number of array dimensions appearing in this expression. The number of dimensions found is entered in the dictionary (DICT) to complete the description of the formal parameter. If this parameter recurs with a varying number of dimensions, a flag is placed in the relevant dictionary entry. A similar method is used to calculate the number of parameters of a formal procedure. (See Appendix A for further details.)

At the end of each procedure, the section of the dictionary corresponding to the procedure is rewritten on SYSUT2 if the description of a formal parameter has been modified using the method described above.

When an array or a procedure (including standard functions and input/output procedures) is used as an actual parameter, a description of the actual parameter is entered in Table PCT. The latter is placed on SYSUT1 (after the intermediate text) when the buffer is full.

The output from Phase 20 is an intermediate text, which is placed on SYSUT1.

On completion of this phase, control returns to the Control Phase of the program.

PHASE 20 ROUTINES

This section contains detailed descriptions of the routines used by Phase 20. The organization is the same as that of the section "Processing Routines" in Phase 10.

BEGPRO ---------- Chart 024

Purpose: To detect the presence of a block, and to update the dictionary.

Called by: Phase 20.

Processing: The next block section of the dictionary has already been stored in DICEXT, and IRNEXT contains the internal reference number of the line opening the block. Detects block if the key of the current line (CURKEY) is equal to IRNEXT. In the latter case, the contents of DICEXT are added to DICT and the various pointers are updated. If the block found is not the last in the program, the next section of the dictionary is read from SYSUT2 and entered in DICEXT. BEGPRO checks with BLIDIC, which indicates the total number of blocks, to see if the current block is the last one.

Routine called: None.

Exit: SDCID.

PDCPRO ---------- Chart 025

Purpose: To analyze the type of the procedure declaration and, if necessary, to add the relevant section to the dictionary.

Called by: Phase 20.

Processing: Finds the procedure identifier in DICT and adds its description to the line in area PRDESC of the output buffer. If the procedure declaration is not the first of the program, adds to the line in the output buffer the internal reference number of the previous declaration. If the procedure contains parameters, reads the list of parameters, together with their description, from DICPRO. Counter NBPRO, initialized to 0 and incremented by 1 at each occurrence of a procedure declaration containing parameters, provides the appropriate dictionary section number. If the procedure body is other than 'CODE', routine PDCPRO enters the contents of DICPRO in DICT and updates the various pointers. NBPRO is saved.

Routine called: None.

Exit: WRTLIN.

PENPRO_____Chart 026

Purpose: To process the ends of the procedures that contain parameters.

Called by: Phase 20.

Processing: SDCID has previously modified the description of the formal parameters specified by 'ARRAY' and 'PROCEDURE' in the procedure declaration.

Routine PENPRO then rewrites the corresponding section of the dictionary on SYSUT2, using as a key the procedure number previously saved during the processing of the procedure declaration.

Routine called: None.

Exit: BKLEND.

SDCID_____Charts 027,028,029,030

Purpose: To search for identifiers in the dictionary and to analyze procedure statements.

Called by: Phase 20.

Processing: Scans each line word by word.

Whenever it finds an identifier:

1. If this identifier is followed by a colon (except in array declarations), it is a label.
2. If the identifier appears in the dictionary:
   a. If it is a procedure identifier present in the left part of an assignment statement, it is replaced by $RTURN.
   b. A new entry is placed in the stack if any of the following types of identifier are found:
      - Formal parameter specified by 'ARRAY' and followed by a left bracket.
      - Formal parameter specified by 'PROCEDURE' and followed by a left parenthesis.
      - Procedure identifier followed by a left parenthesis; the procedure having a formal parameter specified by 'ARRAY' or 'PROCEDURE'.
   c. If an array or procedure identifier appears as an actual parameter, the information concerning this identifier (i.e., number of dimensions, section number, address, etc.) is entered in the stack. (See Table PCT in Appendix A for further details.)
3. If the identifier was not found in the dictionary:
   a. If the delimiter following it is a left parenthesis, the identifier must be the name of a standard function or of an input/output procedure.
   b. In all other cases, the identifier is a label or an ALGOL function name used as an actual parameter.

Whenever it finds a delimiter:

The processing is the same as for a delimiter following an identifier, i.e.:

1. If the stack is not empty, checks the delimiters, as follows:
   a. If the delimiter is a left bracket or a left parenthesis, updates bracket and parenthesis counter.
   b. If it is a comma, calculates the number of dimensions for a formal array, or the number of parameters for a procedure.
   c. A right bracket or a right parenthesis indicates the end of a procedure statement, function desig-

nator, or subscript list. Routine SDCID then checks the last entry in the stack and enters in the dictionary the information required to complete the description of the corresponding formal parameter. This entry is copied into Table PCT. (See Appendix A for further details.)

2. If the stack is empty, takes no action.

Routine called: None.

Exit: WRTLIN.

Chart 023. Overall logic of Phase 20

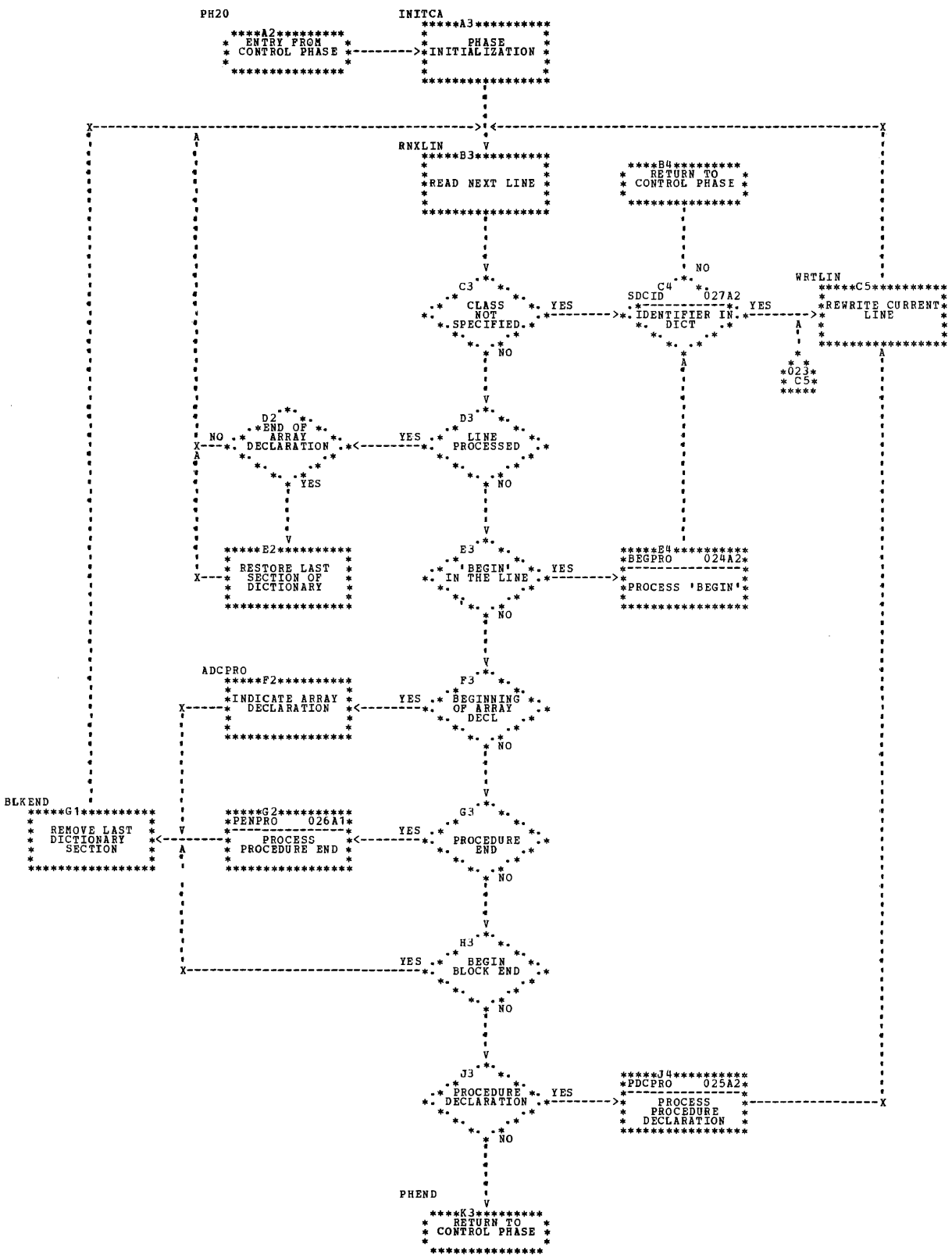


Chart 024. BEGPRO Routine

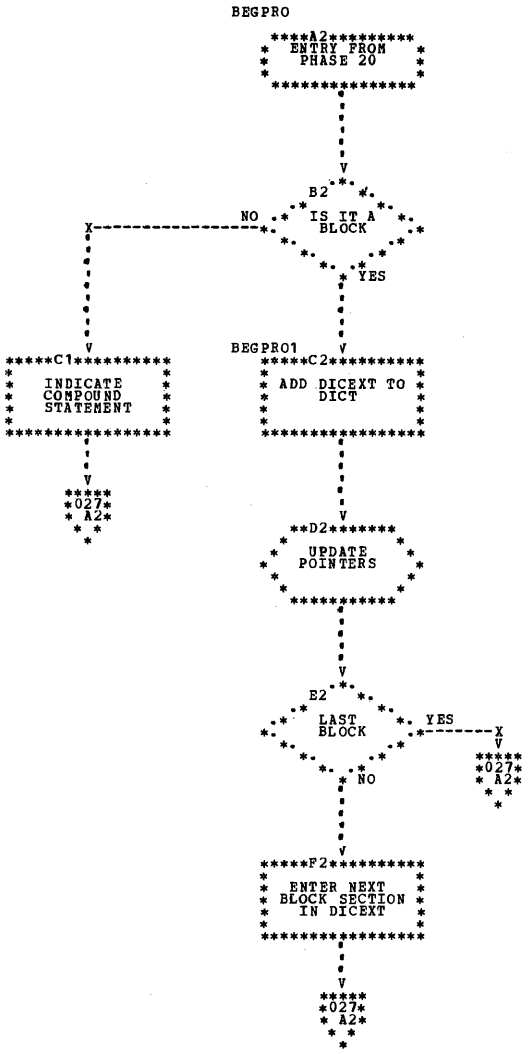


Chart 025. PDCPRO Routine

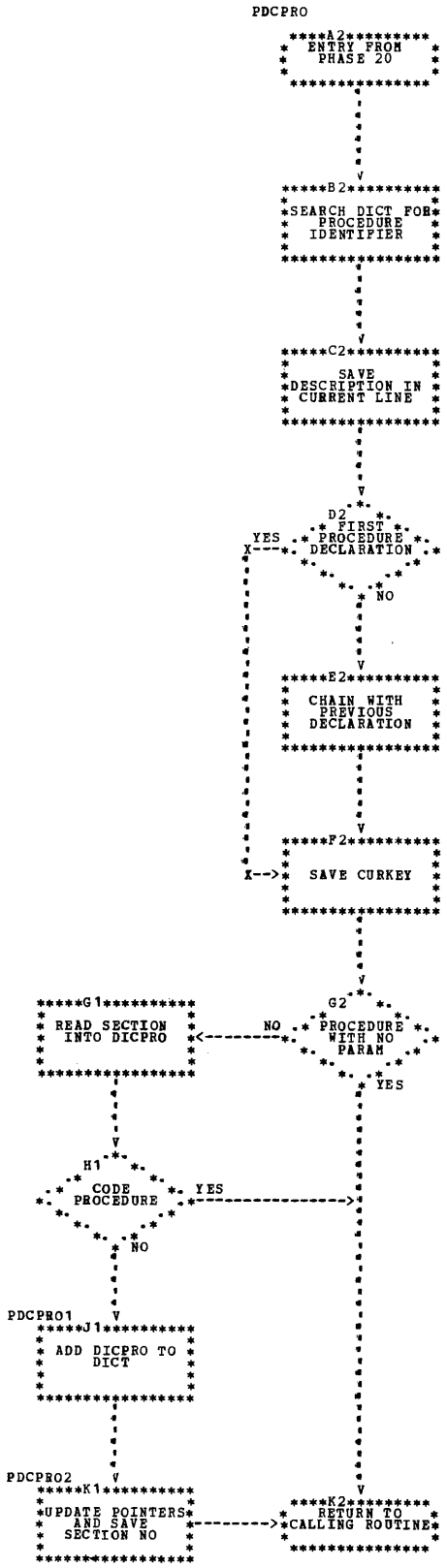


Chart 026. PENPRO Routine

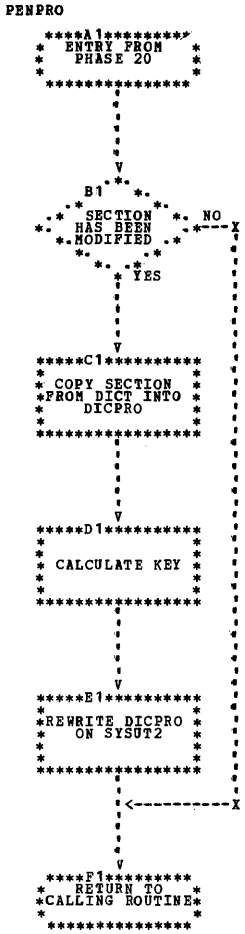


Chart 027. SDCID Routine, Part 1 of 4

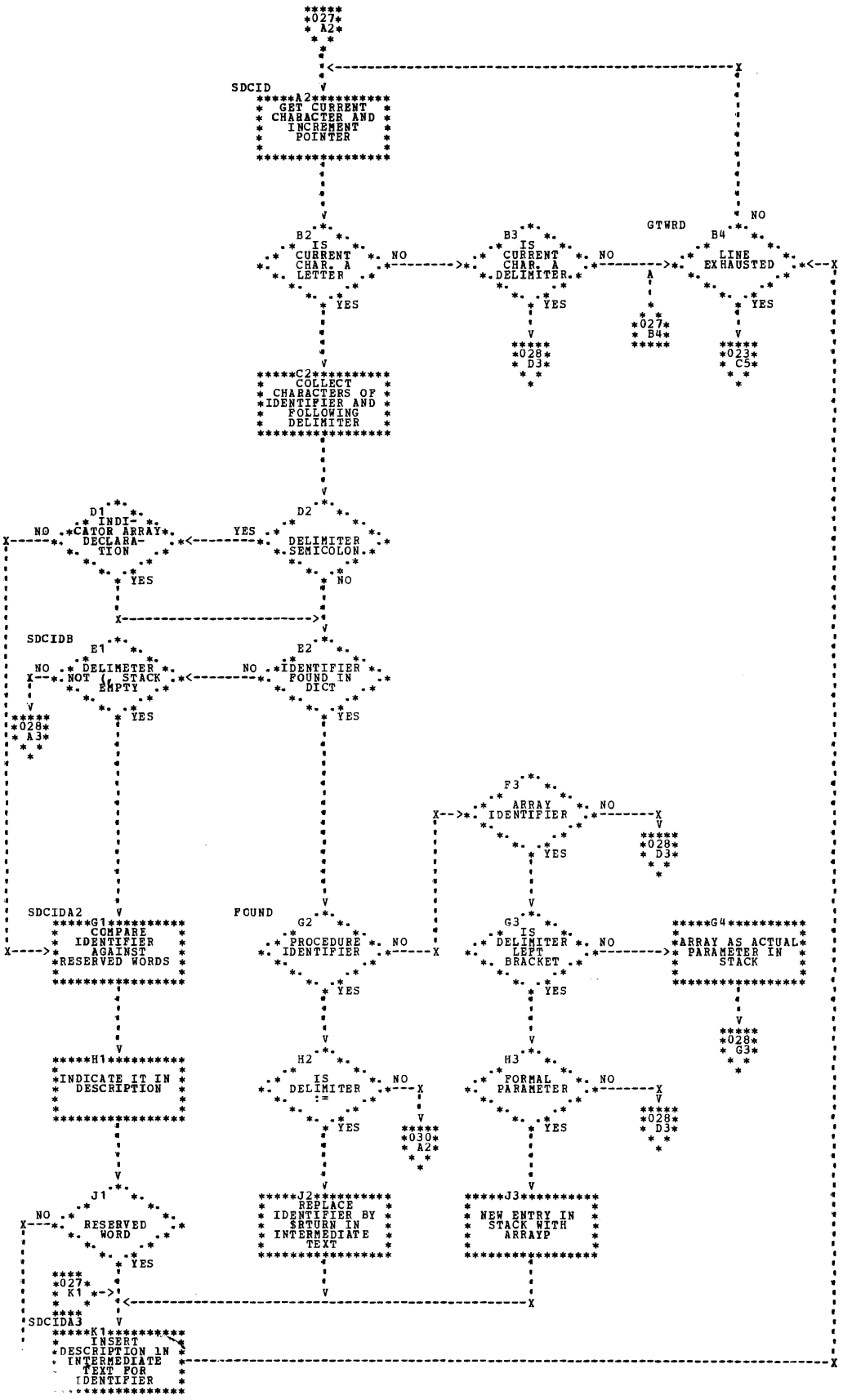


Chart 028. SDCID Routine, Part 2 of 4

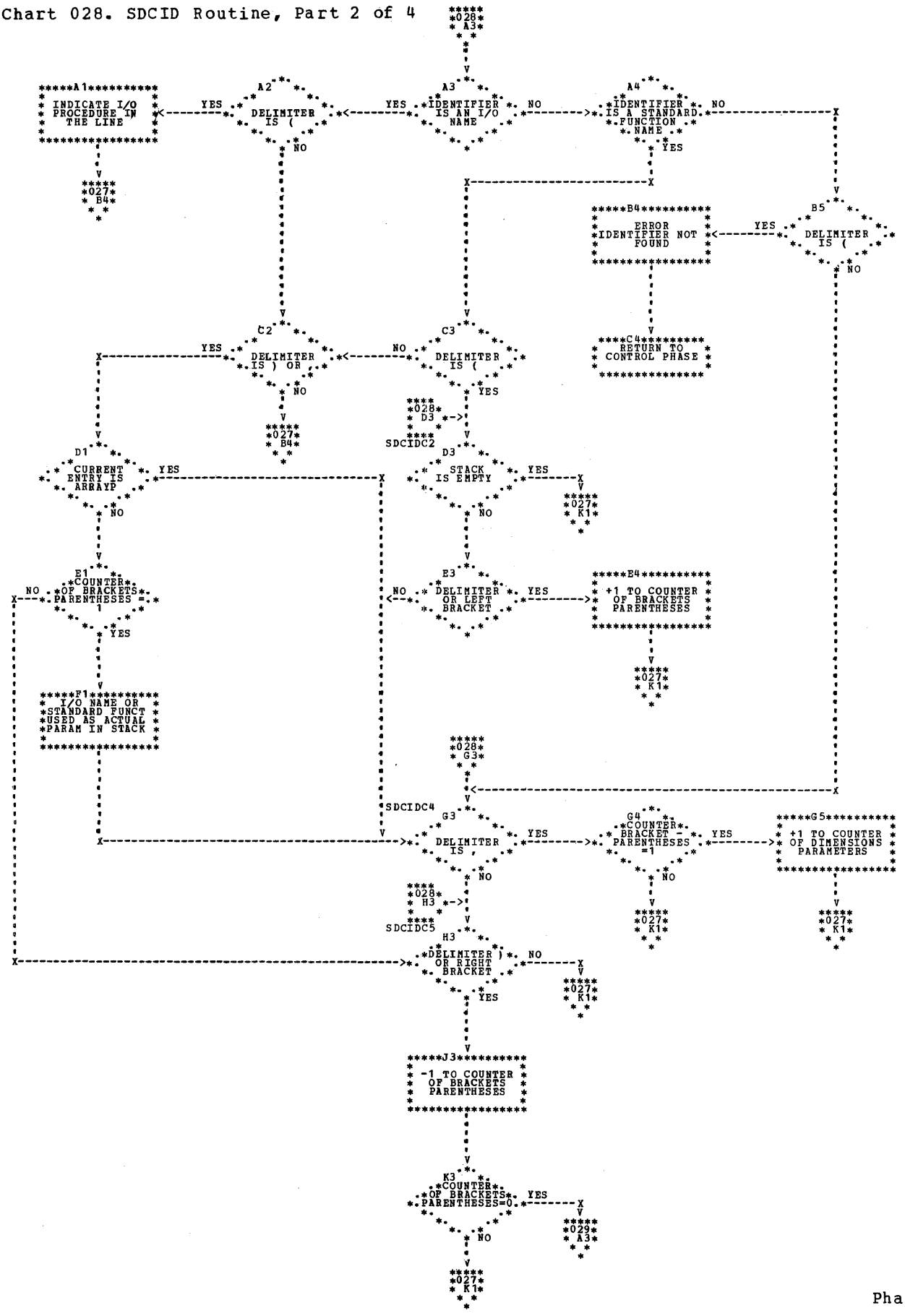


Chart 029. SDCID Routine, Part 3 of 4

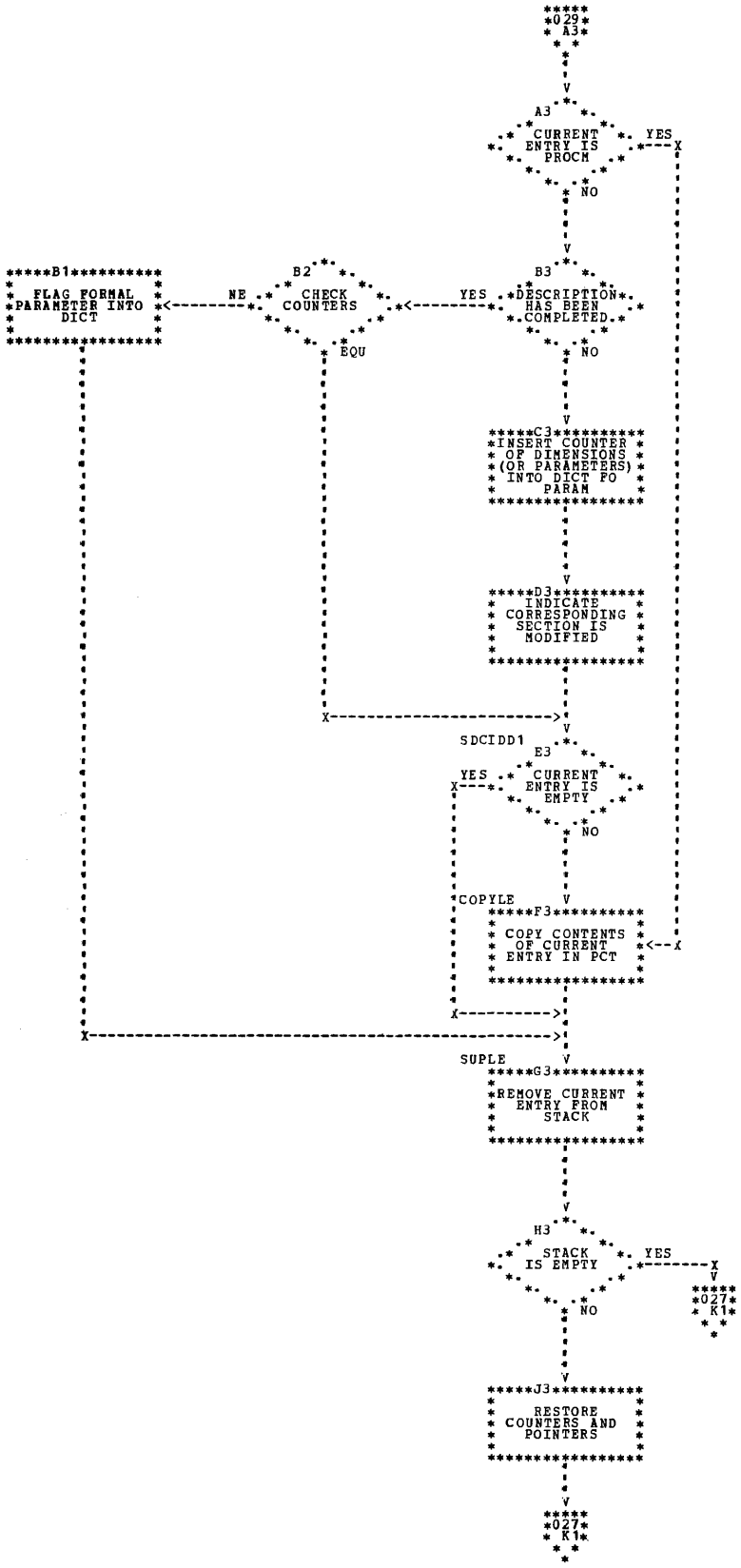
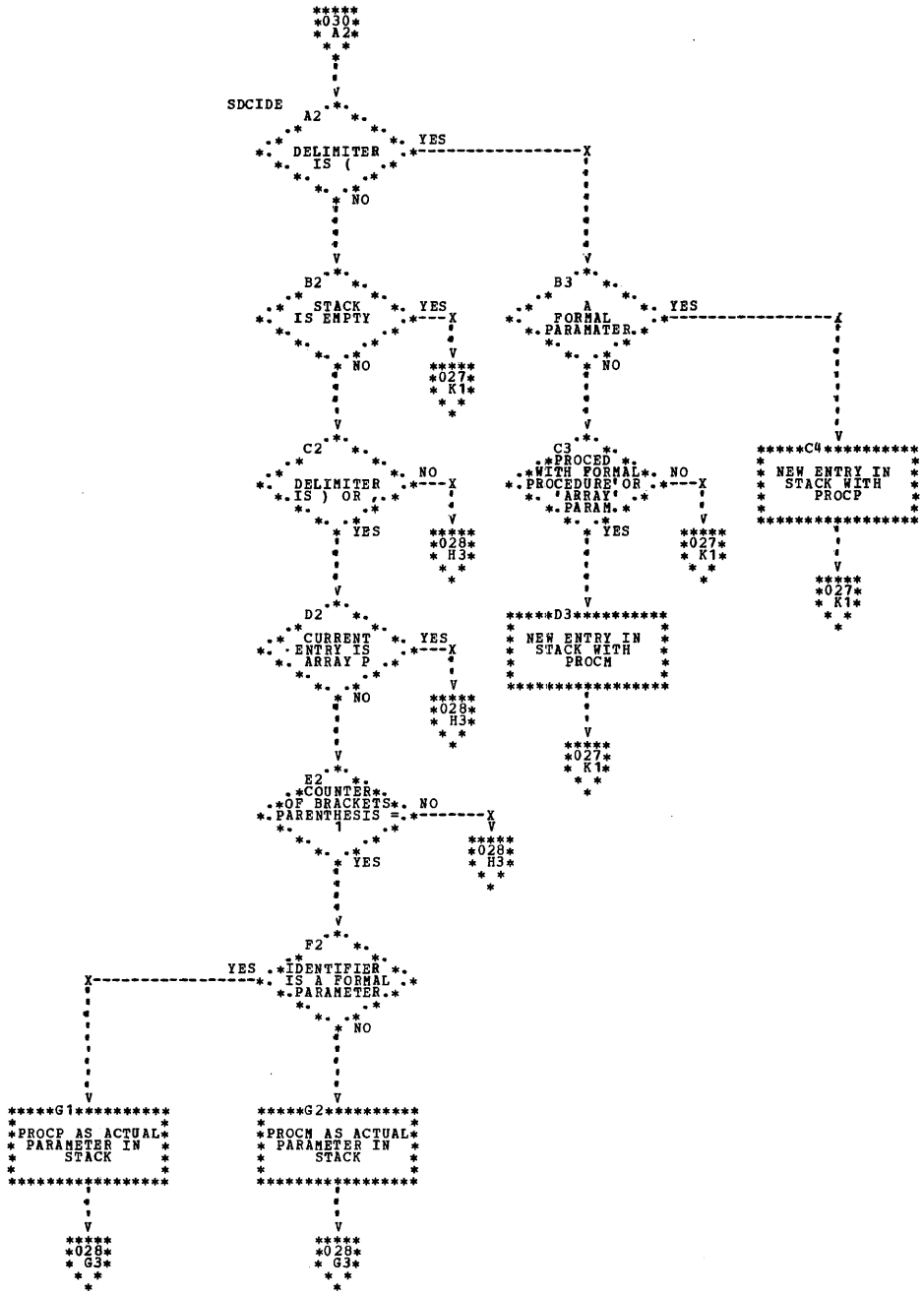


Chart 030. SDCID Routine, Part 4 of 4



PHASES 25 AND 30

Both Phases 25 and 30 are optional; they are called only if certain conditions are present in the program.

PHASE 25

The input to Phase 25 consists of Table PCT on SYSUT1 and the dictionary on SYSUT2. During the initialization of Phase 25, Table PCT is taken from SYSUT1 and placed in main storage. Phase 25 completes the description of formal parameters specified by 'ARRAY' or 'PROCEDURE'. It is called if both of the following conditions are present:

1. The program contains procedures.
2. One of these procedures contains formal parameters specified by 'ARRAY' or 'PROCEDURE' (i.e., Table PCT is not empty).

In the case of formal parameters specified by 'ARRAY', the number of dimensions is determined by examining the corresponding actual parameter.

To process formal parameters specified by 'PROCEDURE', correspondence is first established between each formal parameter and the procedure declaration of the actual parameter. A check is then made to ensure (a) that procedure names passed to the procedure in place of the corresponding formal parameter have the same number of parameters, and (b) that the latter are of the same type and have the same value-part.

The information obtained is entered in the relevant section of the dictionary which is rewritten on SYSUT2.

When Table PCT has been processed, control returns to the Control Phase of the program.

The overall logic of Phase 25 is illustrated in Charts 031 and 032.

PHASE 30

Phase 30, the input to which consists of the intermediate text on SYSUT1 and the dictionary on SYSUT2, converts procedure headings. This phase is called if a procedure declaration is present in the ALGOL program. (Procedure declarations were chained during Phase 20, thus avoiding the necessity of scanning the whole of the intermediate text.)

The following sequence of PL/I statements is generally used in the conversion of ALGOL procedure headings:

1. Declaration of entry-name with the attribute ENTRY
2. PROCEDURE statement
3. Declaration of formal parameters
4. Comment (value-part)

To generate statement 1, Phase 30 uses the section of the dictionary corresponding to this procedure declaration. If the procedure is used as a function, a dummy variable ($RTURN) is created, which has the same data attributes as the procedure identifier. When a procedure declaration is a "precompiled" procedure (see Section 5.4.6. of the ALGOL language manual), statement 1 is not generated.

Statements 2 and 4 are generated in Phase 10. If the procedure body consists of the delimiter 'CODE', statement 2 is deleted and statement 4 is not generated.

Formal parameters are declared by using the section of the dictionary which served to generate statement 1 above. The declaration of formal parameters declared by 'PROCEDURE' is performed by using the information entered in the dictionary by Phase 25.

The effect of the value part is simulated by statement 1 and by the conversion of actual parameters in Phase 40.

The output from Phase 30 is an intermediate text which is placed on SYSUT1.

When all procedure declarations have been converted, control returns to the Control Phase of the program.

PHASE 30 ROUTINES

Phase 30 consists of a main routine, PRPRO, which calls on two other routines, PALGEN and RTNGEN, to perform certain functions.

These three routines are described in detail below.

Phase 30, in addition, uses two utility routines:

GENR, which places various program items in the output buffer and thence on SYSUT1 when the buffer is full.

GENID, which does the same for identifiers.

PRPRO_____Charts 033,034

Purpose: According to the kind and to the number of parameters in the procedure declaration, to generate the various statements used to convert this declaration.

Called by: Control Phase.

Processing: Reads the procedure declaration and finds the description of the procedure. If this procedure contains parameters, the relevant section of the dictionary is entered in DICPRO. If the procedure is not an external (precompiled) one, calls routines PALGEN and RTNGEN in succession. The end of the PL/I PROCEDURE statement generated by Phase 10 is determined by the first semicolon encountered. The various statements generated are chained using NXIRN, which contains the identification number of the next buffer. (See "BUFFER" in Appendix A.) These statements are placed on SYSUT1 immediately after the intermediate text placed there by Phase 10. If the procedure body is the symbol 'CODE', the PROCEDURE statement is deleted. In the case where the procedure has a type, routine PRPRO generates an auxiliary variable ($RTURN) to contain the value given in the ALGOL program to the procedure identifier.

The parameters are grouped according to their specification, as follows:

1. Specified by 'LABEL'
2. Specified by 'REAL', 'INTEGER', or 'BOOLEAN'
3. Specified by 'STRING'
4. Specified by 'SWITCH'
5. Specified by 'ARRAY'
6. Specified by 'PROCEDURE'

Routine PRPRO then generates the corresponding declaration using, in the case of 'ARRAY' and 'PROCEDURE', the information supplied by Phases 25 and 30. In the case of 'PROCEDURE' in particular, routine PRPRO reads the section of the dictionary corresponding to the actual parameter. Routines PALGEN and RTNGEN are called to generate the parameter-attribute-list.

When all the parameter specifications have been generated, the preceding procedure declaration is examined. If all declarations are found to have been processed, control returns to Phase 30.

Routines called: GENID, GENR, PALGEN, RTNGEN.

Exit: Control Phase.

PALGEN_____Chart 035

Purpose: To generate the parameter-attribute-list corresponding to a declaration with the ENTRY attribute.

Called by: PRPO.

Processing: Examines the specifications of the formal parameters and generates an attribute when it finds any one of the following:

- 'REAL' procedure
- 'REAL' array or 'INTEGER' array called by value
- Type procedure called by value
- Arithmetic variable ('REAL' or 'INTEGER')

In all other cases the attribute corresponding to the parameter is empty.

Routines called: GENR, GENID.

Exit: Calling routine.

RTNGEN_____Chart 036

Purpose: To generate the RETURNS attribute when a function is encountered.

Called by: PRPRO.

Processing: Checks the procedure identifier for type; if a type is present generates RETURNS followed by the corresponding data attribute.

Routine called: GENR.

Exit: Calling routine.

Chart 031. Overall Logic of Phase 25, Part 1 of 2

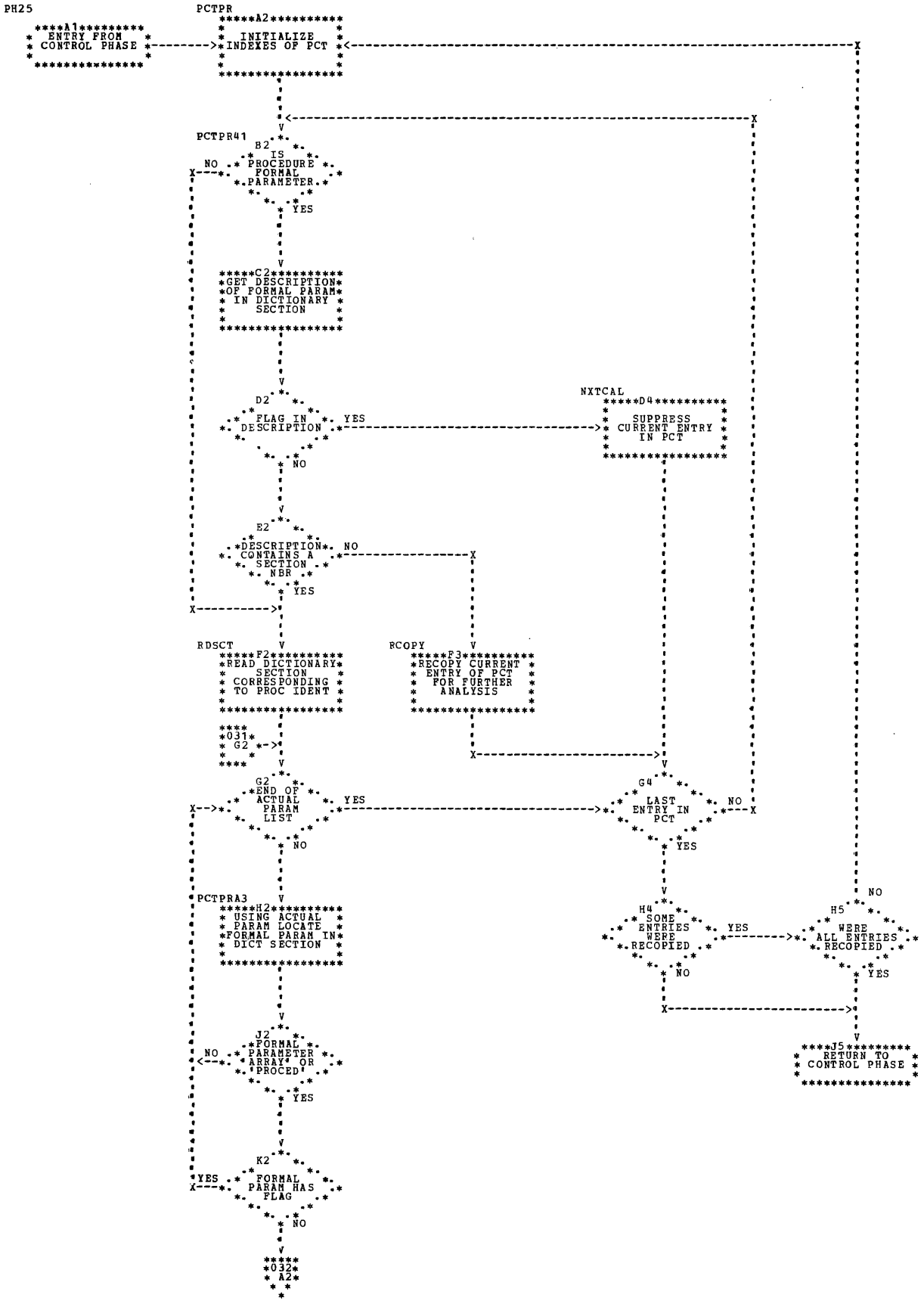


Chart 032. Overall Logic of Phase 25, Part 2 of 2

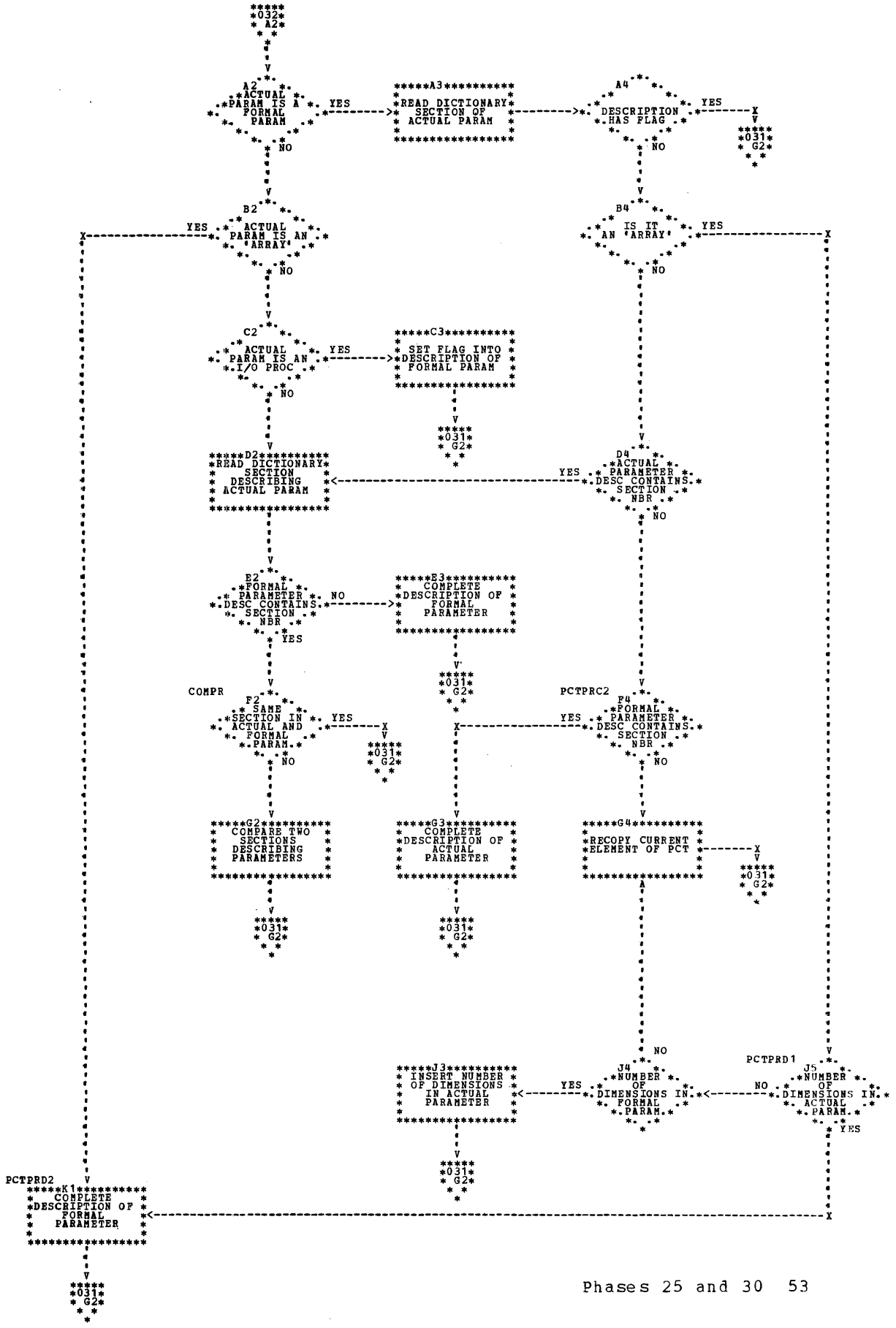


Chart 033. PRPRO Routine, Part 1 of 2

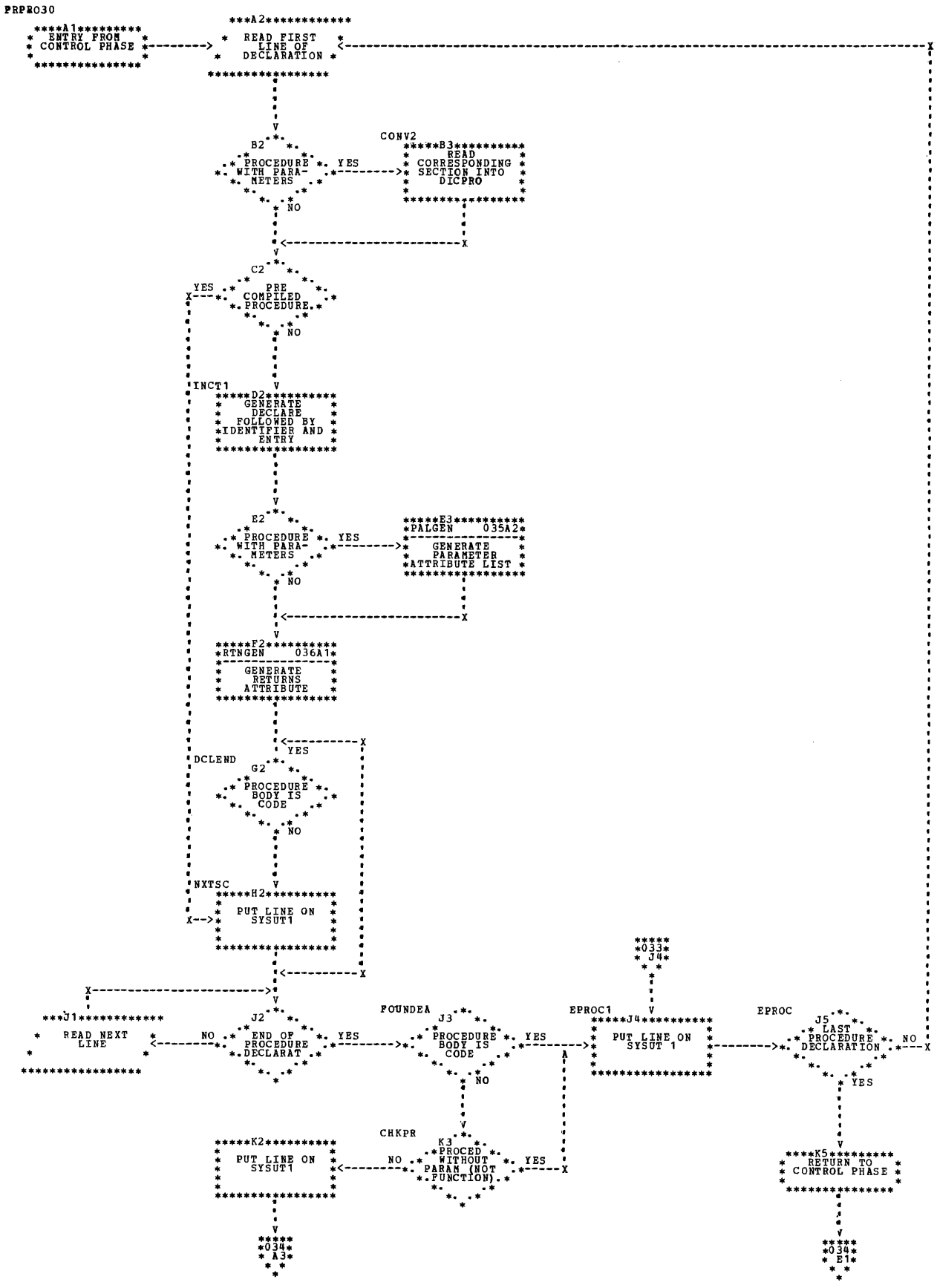


Chart 034. PRPRO Routine, Part 2 of 2

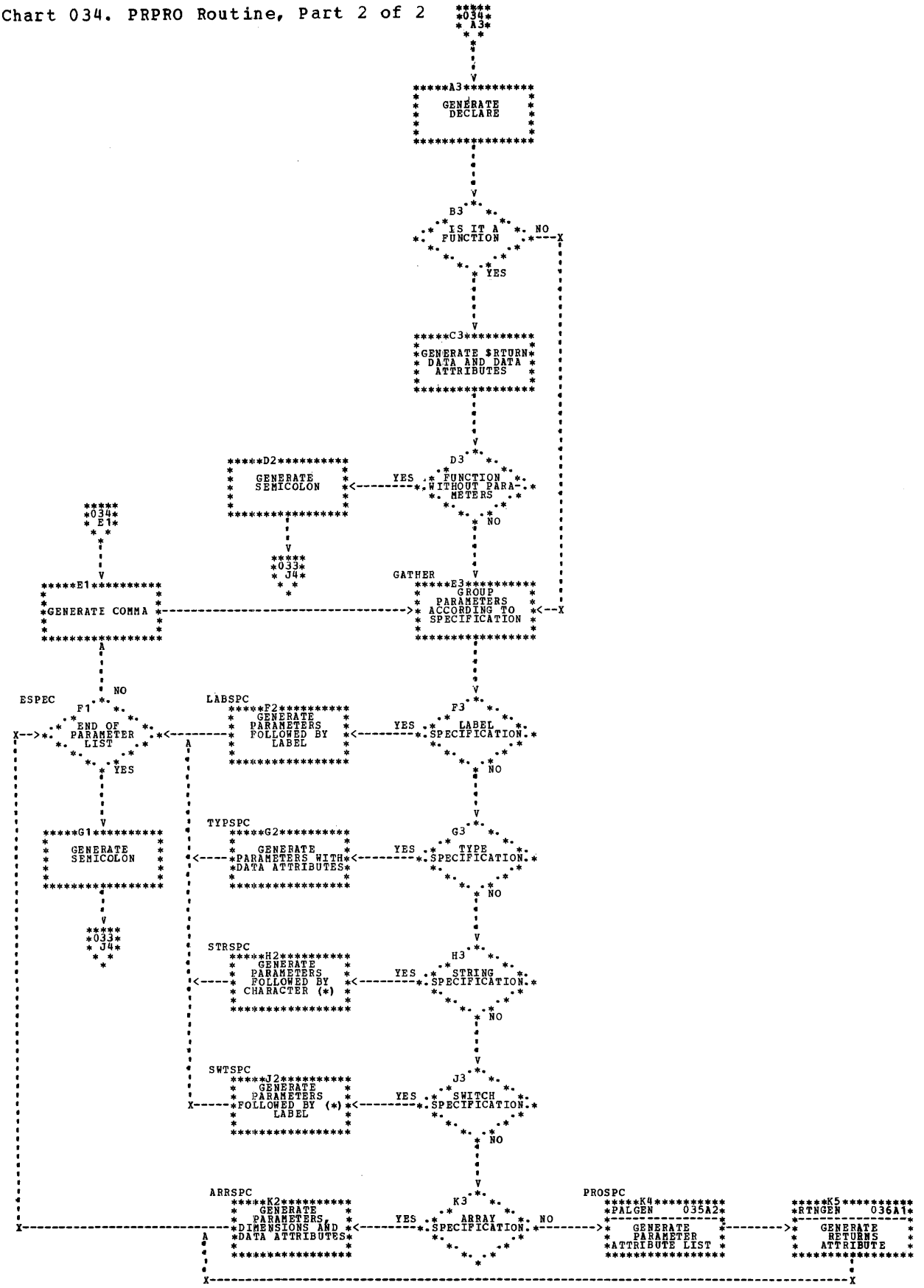


Chart 035. PALGEN Routine.

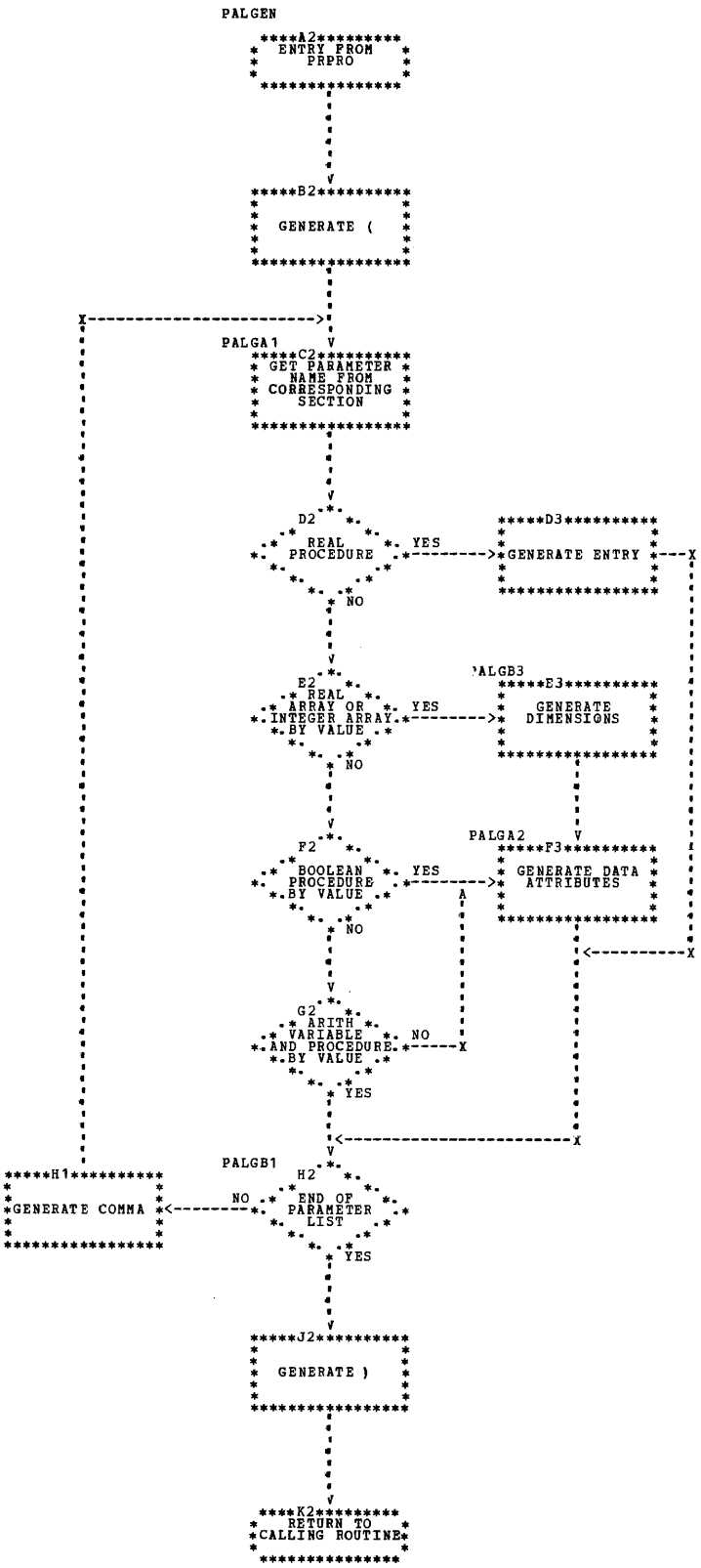
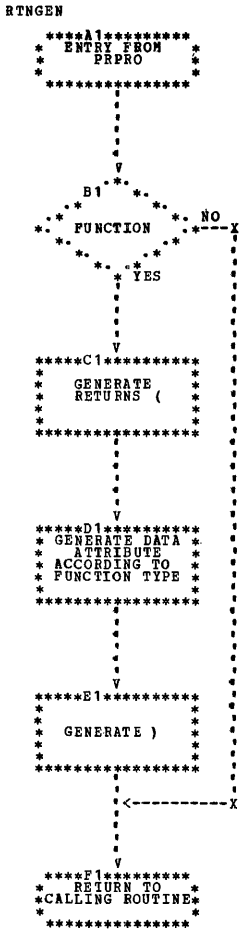


Chart 036. RTNGEN Routine



Phase 40 (see Chart 037) generates the PL/I target program. The input to this phase is the intermediate text on SYSUT1, together with the dictionary on SYSUT2 if the ALGOL program contains procedures (see Phases 25 and 30).

After the initialization of phase parameters, and when processing a main program, Phase 40 generates a PROCEDURE statement with the label MAINPRO.

Executable statements and partially converted declarations (array declarations, for example) are converted by various statement processing routines.

Routine EXPRO is called as required by the statement processing routines, to process arithmetic and boolean expressions.

Phase 40 has to check the type of an arithmetic expression when any of the following are present:

- Assignment statements with left-part variable of type 'INTEGER'
- Subscript expressions
- Actual parameters with corresponding formal parameters specified by 'INTEGER' and called by value

If the arithmetic expression is of type 'REAL', the built-in function FLOOR is used to simulate the rounding in ALGOL. Parentheses are inserted where necessary, because of differences between ALGOL and PL/I as to the relative priorities of certain operators. Simulation of the ALGOL operators 'EQUIV', 'IMPL', and '/' requires sophisticated scanning by EXPRO of the expression, using a stack and the relative priorities of the operators.

Conversion of input/output procedures depends on the kind and type of the actual parameters. If a statement cannot be converted, or if conversion is doubtful, the relevant line in the output listing is identified by a number (see Appendix F of the ALGOL-to-PL/I language conversion manual).

If the source program is found to contain a conditional expression, the functions $CONAEX and/or $CONBEX are generated at the end of the converted program.

Output from Phase 40, i.e., the target program, is in the form specified by the user in his EXEC control card.

On completion of Phase 40, control returns for the last time to the Return Procedure of the Control Phase.

PHASE 40 ROUTINES

The routines used by Phase 40 fall into three categories:

- Statement processing routines
- Expression processing routine (EXPRO)
- Utility routines

The first two categories are described in detail and illustrated by flowcharts, as in previous sections of the manual; the utility routines are described more summarily.

STATEMENT PROCESSING ROUTINES

ARPRO ---------- Chart 038

Purpose: To process array declarations.

Called by: Phase 40.

Processing: Phase 10 has already converted the part of each array declaration which does not need a dictionary reference. Calls routine EXPRO to convert the bound-pair-list, each bound being of type 'INTEGER'. Using a left bracket, an entry is placed in the stack. The end of a bound-pair-list is detected when a right bracket deletes this entry.

The converted lines are transmitted, without modification, to the output buffer. A semicolon found in a converted line indicates the end of an array declaration.

Routines called: EXPRO, PRTLIN.

Exit: Calling routine.

ASPRO Chart 039

Purpose: To process assignment statements.

Called by: Phase 40.

Processing: Begins by processing the left-part variable. If it is a subscripted variable, calls routine EXPRO to convert the corresponding expression, which must be of type 'INTEGER'. Using the left bracket, an entry is made in the stack.

Then processes the right-hand part of the statement. The symbol := is used to place an entry in the stack. The type of the expression is obtained from the type of the left-part variable.

Routines called: EXPRO, GEN40, PUTID.

Exit: Calling routine.

FORPRO Chart 040

Purpose: To process ALGOL 'FOR' statements.

Called by: Phase 40.

Processing: Converts the controlled variable and examines the for-list, giving the type of the controlled variable to routine EXPRO.

In the case of STEP-UNTIL element, issues a warning message if the expression following 'STEP' or 'UNTIL' contains variables.

In the case of the WHILE element, issues a warning message if the expression preceding 'WHILE' contains variables. The expression following 'WHILE' is always of type 'BOOLEAN'.

Routine called: GEN40, PUTID.

Exit: Calling routine.

GOTOPRO Chart 041

Purpose: To process 'GO TO' statements.

Called by: Phase 40.

Processing: 'GOTO' is followed either by a left parenthesis, an 'IF' clause, a label, or a switch identifier. GOTOPRO converts these as follows:

- Left parenthesis: ignores it.
- 'IF' clause: calls IFPRO to process it.
- Label: converts it.
- Switch identifier: calls EXPRO to convert the subscript expression.

A designational expression that cannot be converted is scanned to determine the end of the statement, and a warning message is issued.

Routines called: EXPRO, IFPRO, PUTID, GEN40.

Exit: Calling routine.

IFPRO Chart 042

Purpose: To process 'IF' clauses in designational expressions and 'IF' statements.

Called by: Phase 40, GOTOPRO.

Processing: Generates IF in the output buffer. Calls EXPRO to process the boolean expression following 'IF'. The corresponding 'THEN' ends the expression. If the count of 'IF's is zero, the 'IF' clause is not a part of a designational expression and IFPRO returns control to Phase 40. Otherwise control returns to GOTOPRO.

Routines called: EXPRO, GEN40, PUTID.

Exit: Calling routine

IOPRO Chart 043

Purpose: To process input/output statements.

Called by: Phase 40.

Processing: The internal reference number of the input/output procedure appears in PRDESC, in the input buffer; it is used as a pointer to the various sequences of instructions that process the input/output statements.

The correspondence between the ALGOL data-set-number and the PL/I file-name is given as follows:

```
 0 --  SYSIN
 1 --  SYSPRNT
 2 --  ALGLDD02
 3 --  ALGLDD03
   .
   .
15 --  ALGLDD15
```

The DD name is used as the file-name in the PL/I target program. If the data-set-number is not an integer constant, it is not converted.

The dummy variables used to convert the input/output procedures are declared at the end of the converted program, if necessary.

If the data-set was created by another program, or if it stands as the system input data-set, it is the user's responsibility to verify that the data items are in accordance with the PL/I specifications.

The SYSACT control procedure is converted only in certain cases. (For further details on this point, refer to Appendix C of the ALGOL-to-PL/I language conversion manual.)

Routines called: EXPRO, GETWORD.

Exit: Calling routine.

PRPRO Chart 044

Purpose: To process procedure statements.

Called by: Phase 40.

Processing: First, generates CALL statement followed by the procedure identifier. If the procedure contains parameters, calls routine EXPRO (entered at entry-point EXPROL3) to process the parameter list. The end of the parameter list is identified by a right parenthesis of level 0.

Routines called: EXPRO, GEN40 PUTID.

Exit: Calling routine.

EXPRESSION PROCESSING ROUTINE

EXPRO Charts 045,046,047,048,049

Purpose: To analyze the expressions appearing in the various ALGOL statements and to convert the argument list of function designators or of procedure statements.

Called by: All Phase 40 routines.

Processing: When the type of the expression is already known, it is transmitted directly to routine EXPRO. This is so, for example, in the case of subscript expressions, right part of an assignment statement, etc.

Stack

Routine EXPRO uses a stack to determine the structure of the expression. The precedence of operators and of delimiters is the following:

```
 1     **
 2     '/'  /  *
 3     +  -
 4     =  <  >  <=  ¬=  >=
 5     ¬
 6     &
 7     |
 8     'IMPL'
 9     'EQUIV'
10     ELSE*
11     THEN*
12     ¢  'IF'  (  (*  [
       'THEN'  'ELSE'  'WHILE'
       'STEP'  'UNTIL'
       :  )  ]  ,  :=  ;
```

Certain items appear only in the stack:

ELSE* denotes the delimiter 'ELSE' in a conditional expression: the top item in the stack is 'IF'.

THEN* denotes the delimiter 'THEN' in a conditional expression: the top item in the stack is 'IF'.

(* indicates the beginning of an parameter list.

¢ indicates the beginning of an expression.

A new level is created in the stack for the following items:

'IF' ((* [

Conversion Actions

1. Exponent part: If the separator ' follows a digit, it is converted to E and if not, to 1E.

2. Transfer from 'REAL' to 'INTEGER': Except for boolean expressions in 'IF' clauses, the components of arithmetic expressions must be of type 'REAL' or 'INTEGER'. An arithmetic expression is of type 'REAL' if it contains any of the following items:

   - Variable of type 'REAL'
   - The operator /
   - One of the delimiters ' or .
   - The operator ** when the exponent is not an unsigned integer

3. Conditional expressions: According to the rules of ALGOL syntax, the form of a conditional expression is:

   'IF' B 'THEN' X1 'ELSE' X2

   where: B is a boolean expression

   X1 is a simple expression

   X2 is an expression

   The name created by the LCP, which simulates a conditional expression, depends on the type of expressions X1 and X2. Expression X1 is of type 'BOOLEAN' if it contains any of the following items, except in 'IF' clauses:

   - Variable of boolean type
   - Logical values
   - Relational operators
   - Logical operators

4. Logical operator: Parentheses are inserted to eliminate the differences in priorities.

5. Operators 'IMPL', 'EQUIV', and '/': These operators are simulated by PL/I operators, parentheses, and built-in functions. If the second operand of the operator / (the divisor) is of type 'INTEGER', it is converted to (1E0*divisor) which has a floating point value.

6. Multiple assignment: The last entry in the stack and the current delimiter are := and := respectively. The first assignment operator is converted into a comma.

7. Actual parameter list: The descriptions of formal parameters corresponding to the procedure identifier are obtained from sections, of which the numbers have been determined in Phase 25 if the procedure identifier is a formal parameter. Specifications are used for the conversion of actual parameters. Calls by value are simulated by inserting parentheses, or sometimes by using the built-in function FLOOR.

Routines called: COPYIA, EXPEN, GEN40, GETWORD, PUTID.

Exit: Calling routine.

UTILITY ROUTINES

COPYIA -- fills the output buffer, mixing together the characters in the intermediate area and the characters in the auxiliary table. Each character in the stack and each character in the auxiliary table has a pointer associated with it. These pointers give the order in which the characters will be mixed.

The following example shows how COPYIA works during the translation of the ALGOL statement:

```
A:=B+C**2**I;
```

The LCP must add parentheses to the translated statement to make sure that it is evaluated in the same way. The translated PL/I statement in the output buffer is:

```
A=B+(C**2)**I;
```

The text in the intermediate area, the contents of the stack, and the auxiliary table are illustrated below. Beneath each character in the intermediate area appears the value of the pointer when pointing to it.

Intermediate Area

```
B+C**2)**I
123456789...
```

Stack

:=	0
+	2
**	6

Auxiliary Table

(	2

The left parenthesis in the auxiliary table must be added to the characters in the intermediate area. Its value of 2 shows that it must be added after the + sign, which has value 2 in the intermediate area.

EXPEN -- examines a converted expression in the intermediate area. If the type of the expression has been modified, i.e., if the evaluation of an expression gives a result of type 'REAL' and if this value is to be given to an integer variable, the transfer involves rounding in ALGOL, but not in PL/I; it is simulated by the PL/I built-in function FLOOR.

Places the sequence +.5) in the intermediate area, and adds the sequence FLOOR(to the output text.

GEN40 -- generates an item in the output buffer.

PUTID -- generates an identifier, which is placed either:

- In the output buffer if ISW1 = 0
- In the intermediate area if ISW1 = 1

GETWORD - checks the next character, and

1. It is converted: enters the converted part in the intermediate area, checks next character, etc.
2. It is not converted:
   a. If it is a digit, sets the digit indicator and recopies the digit in the intermediate area; then checks next character, etc.
   b. If the next character is a letter, recognizes it as an identifier.

GETWORD has two exits: through one, it transmits the delimiter and the value of the digit indicator; through the other, it transmits the identifier, together with its description.

PRTLIN -- prints a line when full, output being as specified by the user (option DECK or NODECK).

Chart 037. Overall Logic of Phase 40

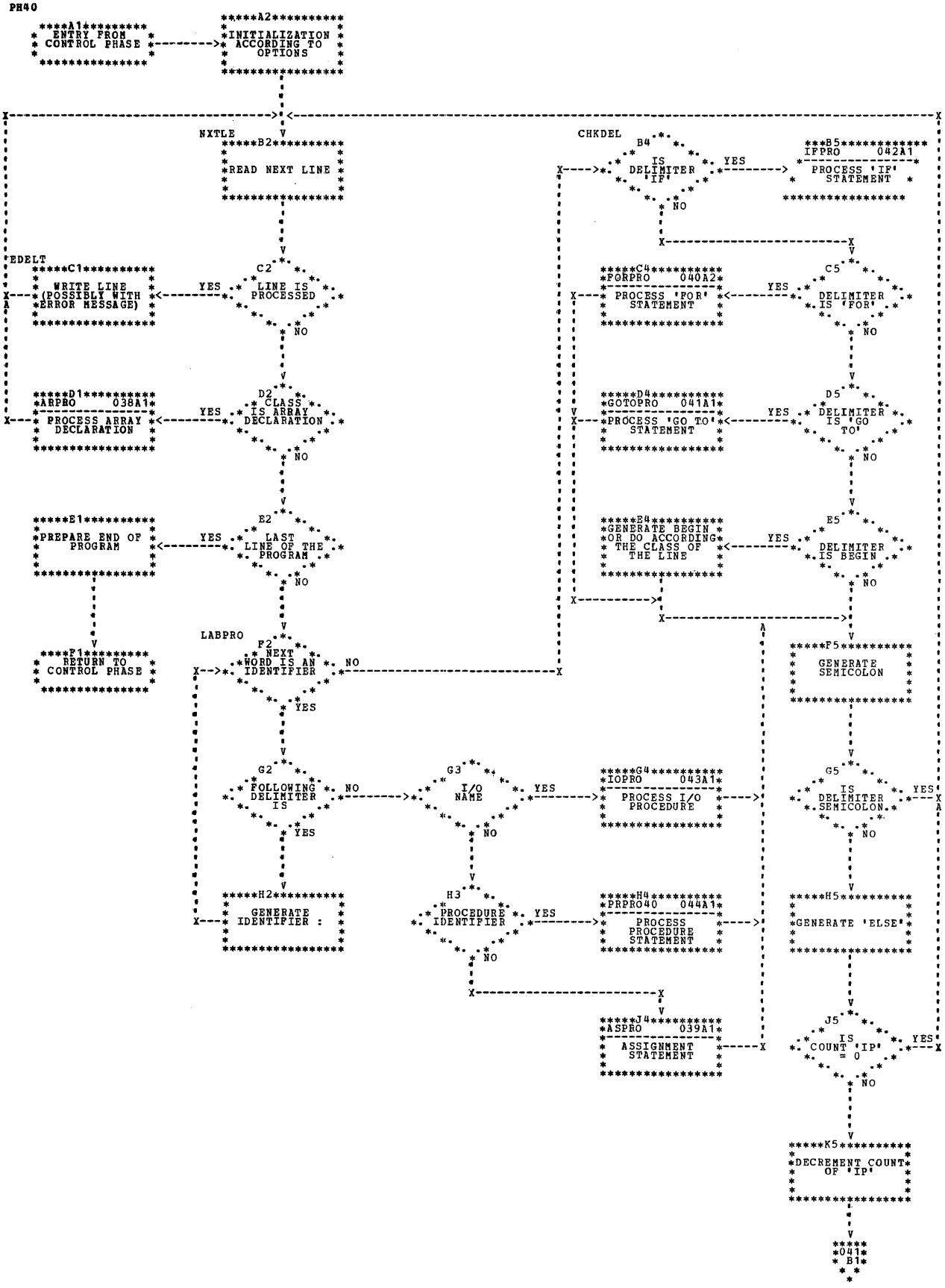


Chart 038. ARPRO Routine

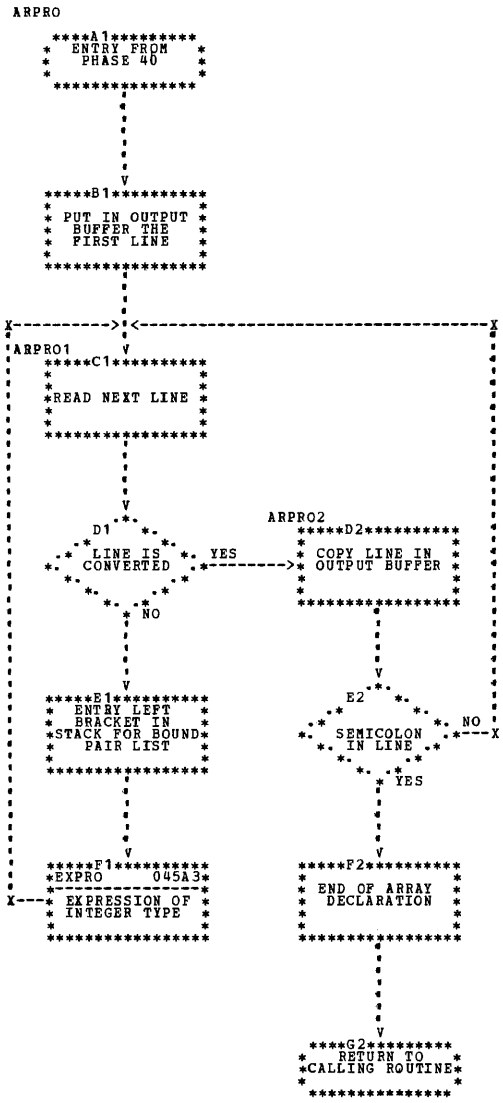


Chart 039. ASPRO Routine

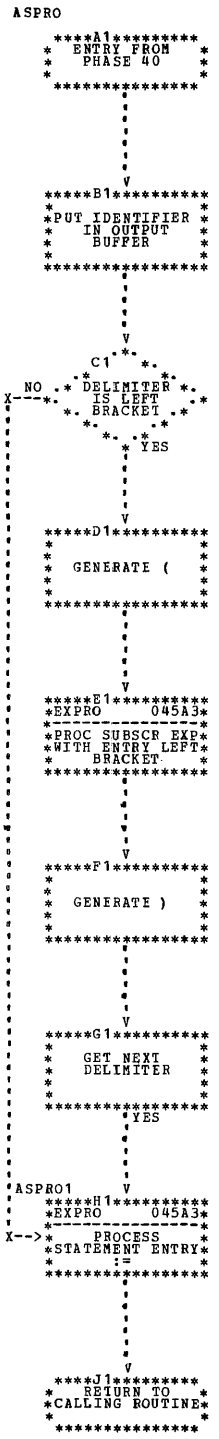


Chart 040. FORPRO Routine

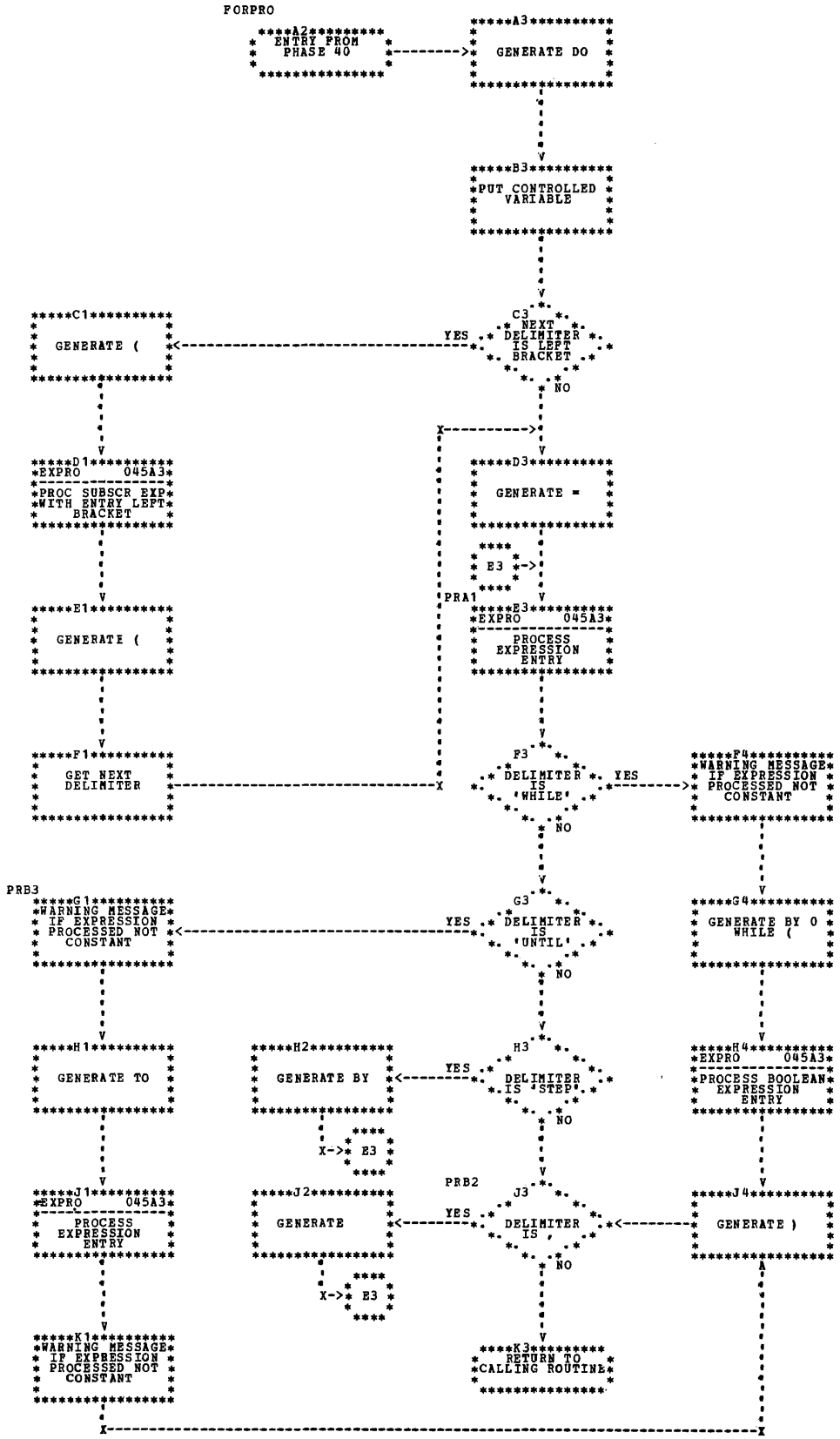


Chart 041. GOTOPRO Routine

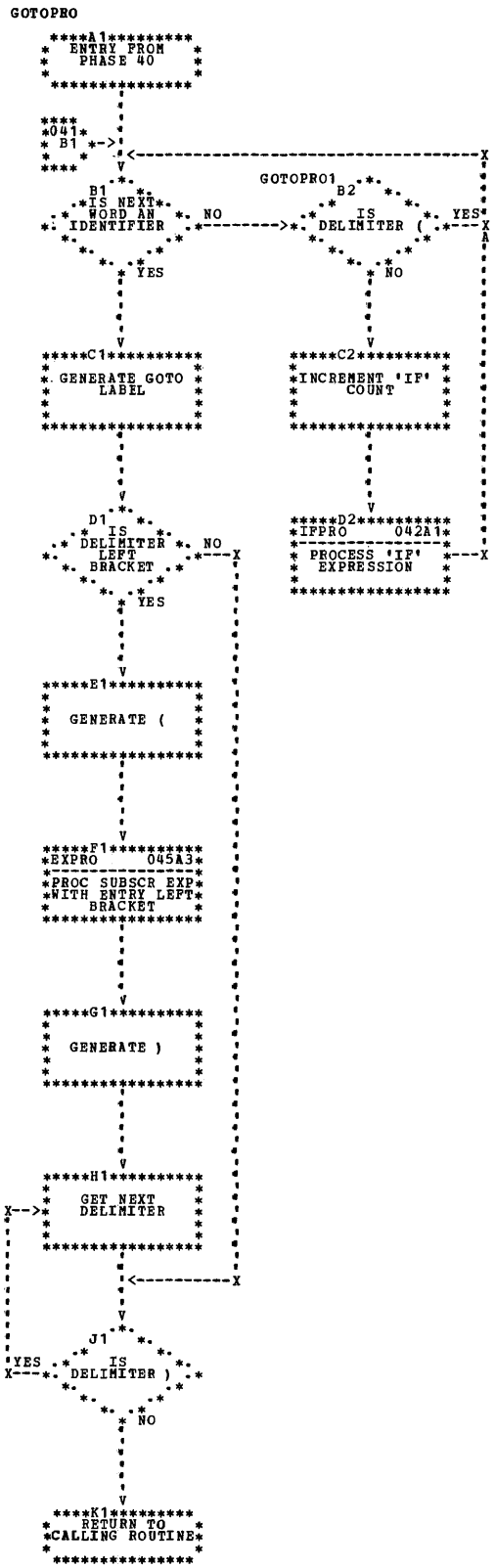


Chart 042. IFPRO Routine

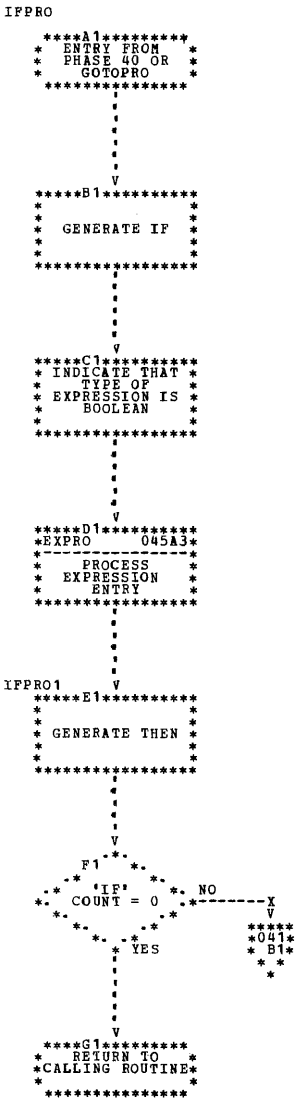


Chart 043. IOPRO Routine

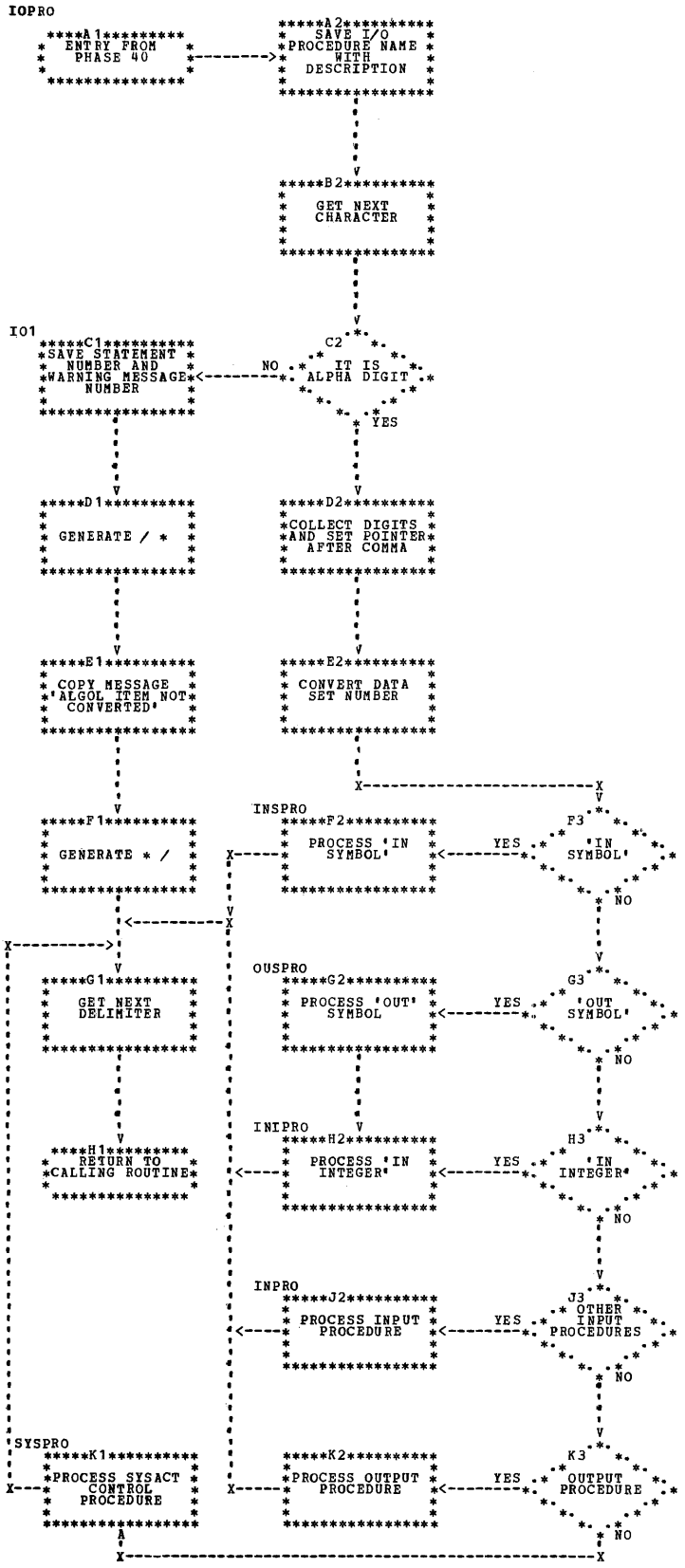


Chart 044. PRPRO Routine

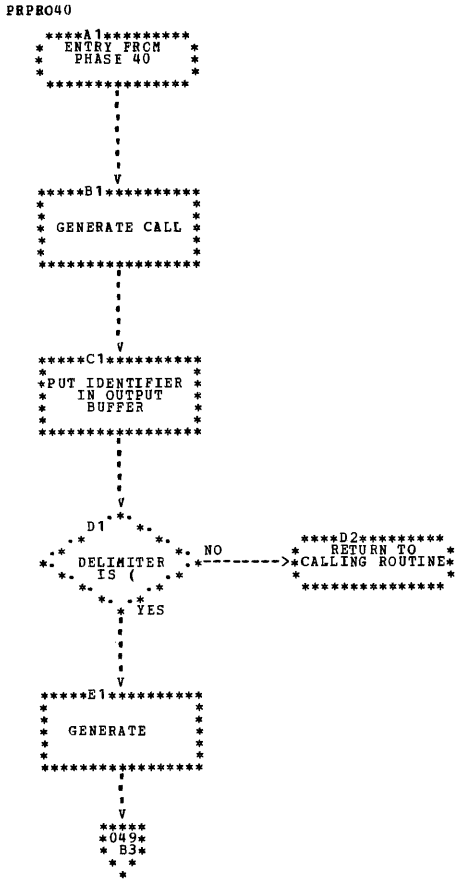


Chart 045. EXPRO Routine, Part 1 of 5

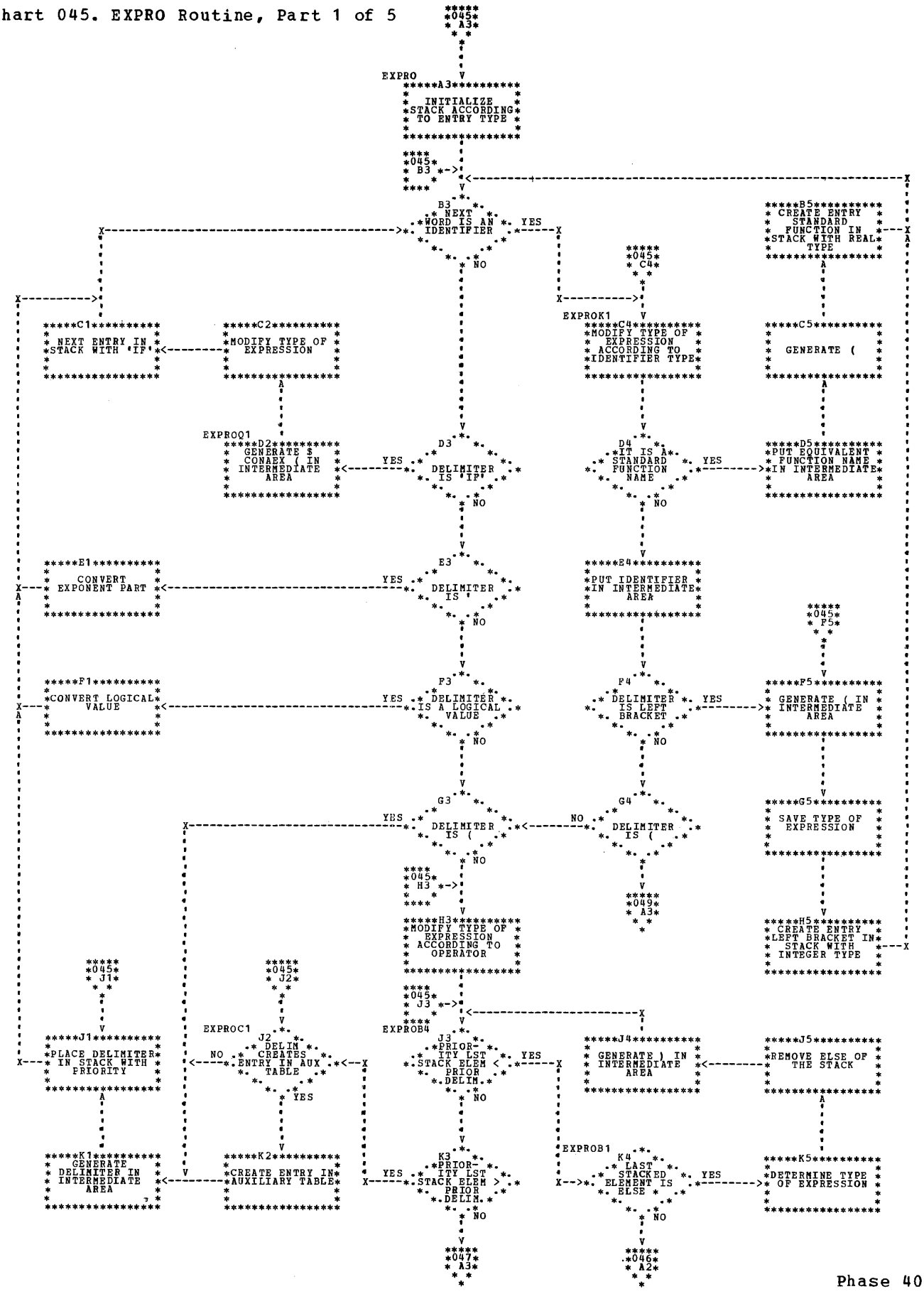


Chart 046. EXPRO Routine, Part 2 of 5

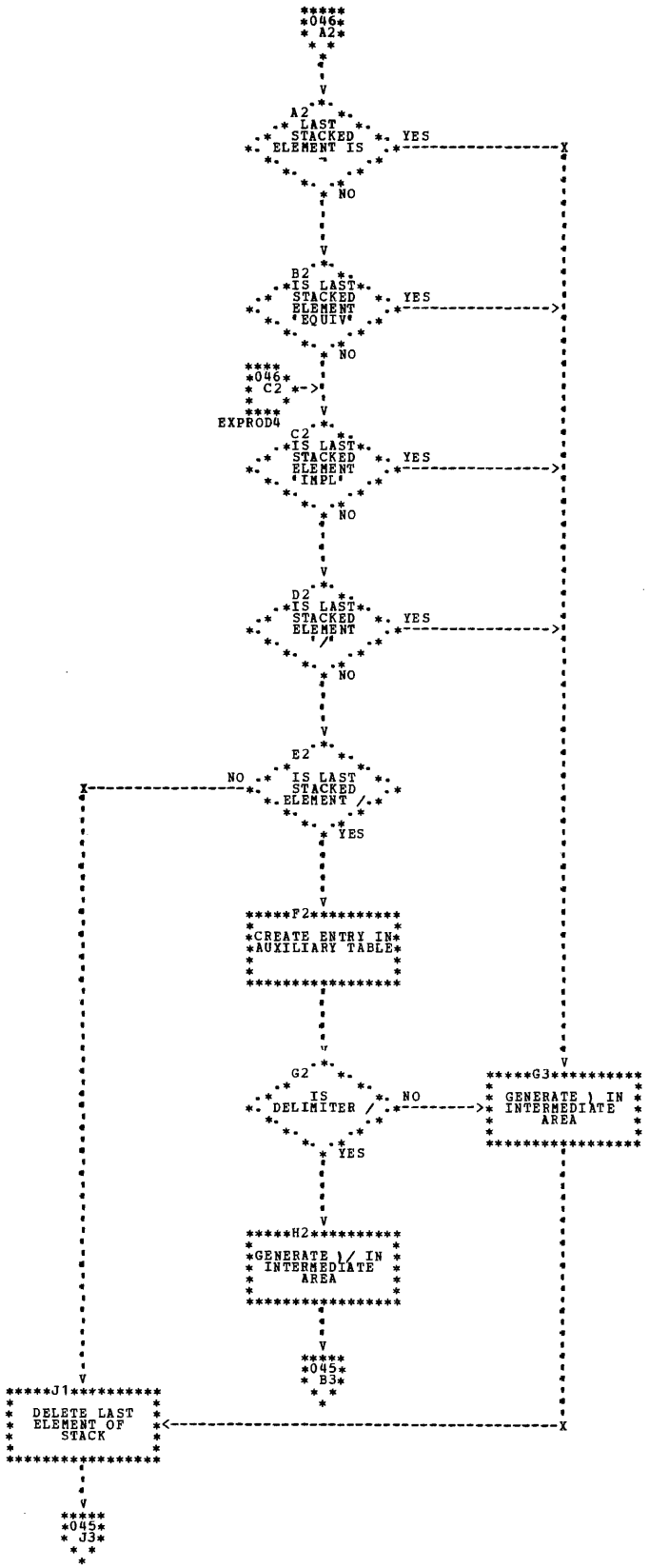


Chart 047. EXPRO Routine, Part 3 of 5

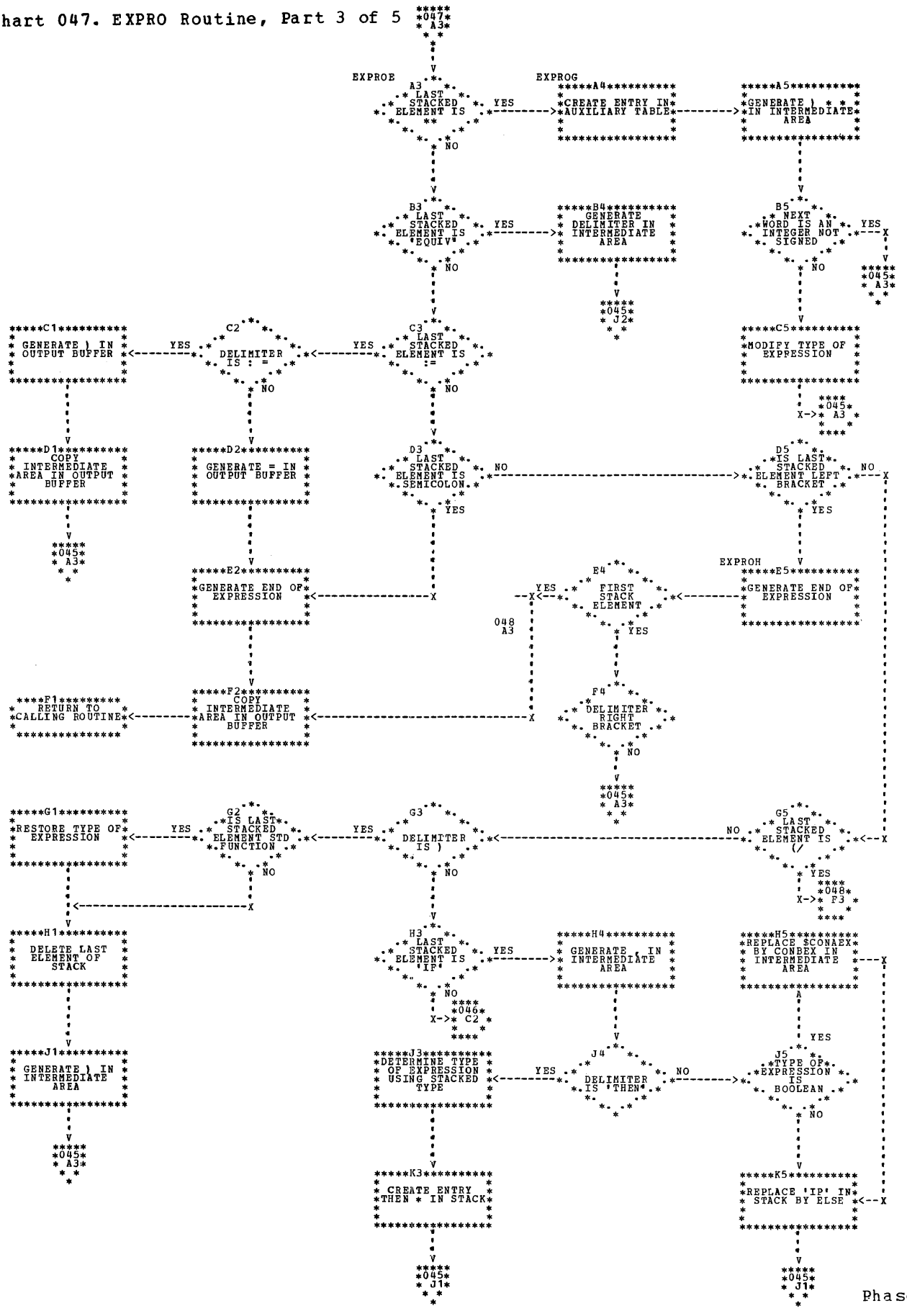


Chart 048. EXPRO Routine, Part 4 of 5

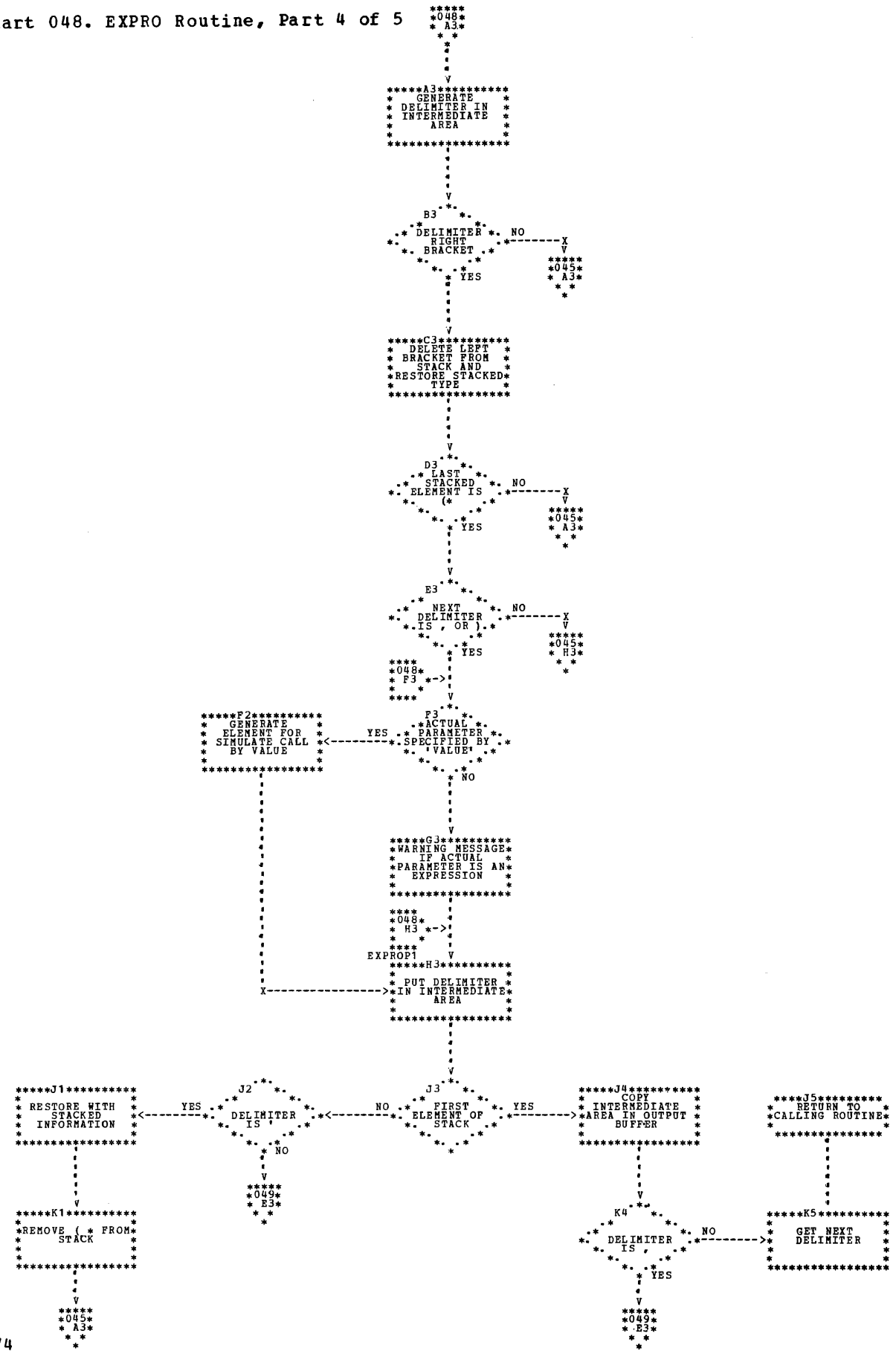
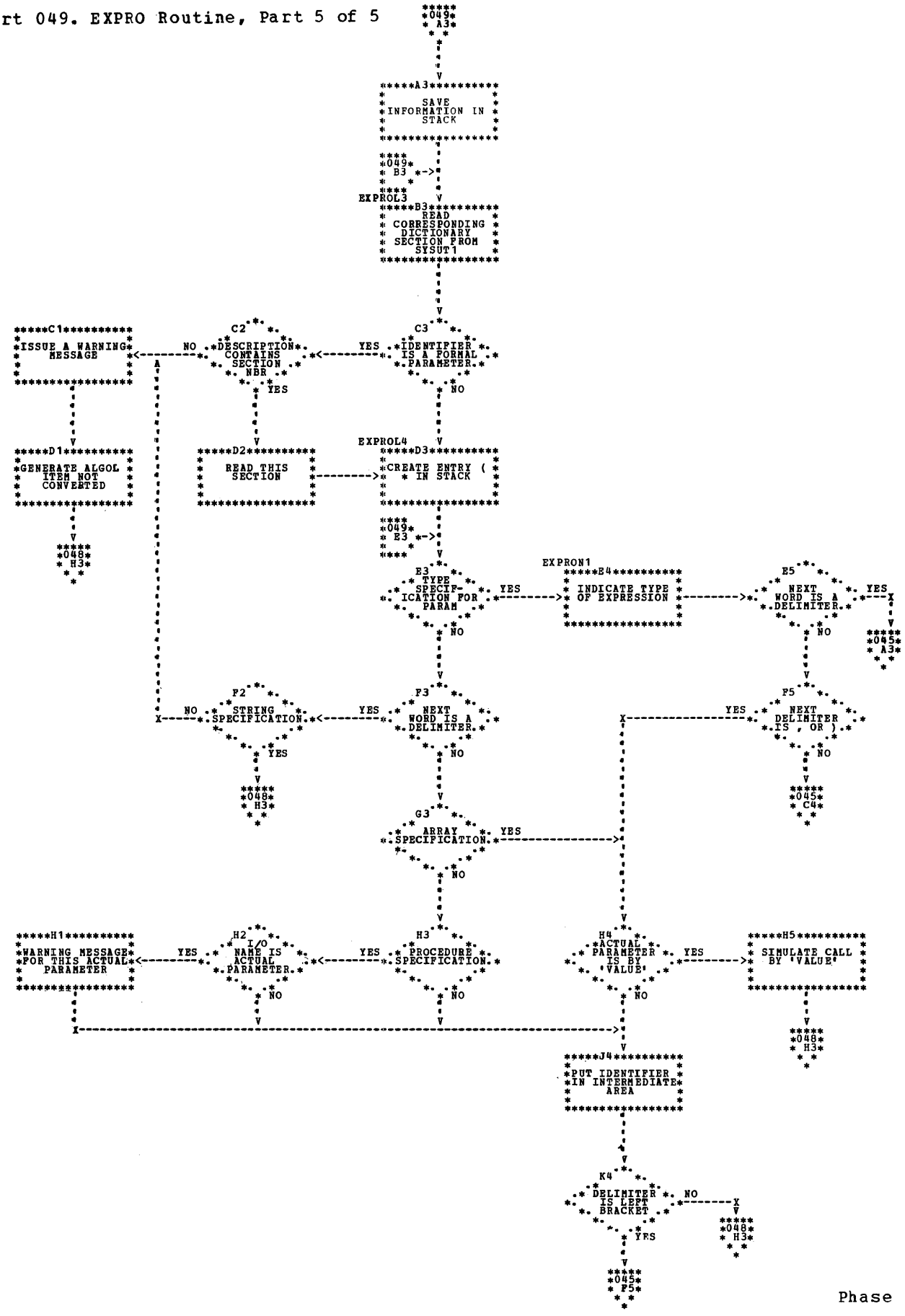


Chart 049. EXPRO Routine, Part 5 of 5



APPENDIX A: TABLES

This appendix contains detailed descriptions of the main tables used by the LCP.

They are:

- The BUFFER
- The Dictionary
- Table PCT
- The communication area COMAREA

BUFFER

BUFFER is an intermediate output area for all phases, placed on SYSUT1 whenever it is completed. The lay-out of BUFFER (not to scale) is as follows:

NCAR -- contains a count of the characters entered in ELEM1.

PRDESC -- (not used by Phase 10) contains a procedure identification: type of the procedure, and in the case of input/output procedures, identification of the procedure.

RMES -- contains the number of the error message concerning the current buffer contents.

AREF -- identifies the card containing the ALGOL statement which is being processed.

NXIRN -- contains the identification number of the next set of information place in BUFFER.

LST -- contains three characters:

First character indicates the contents of BUFFER, as follows:

- D -- block-end
- F -- program end
- C -- 'BEGIN' (in the case of the beginning of a compound statement, Phase 20 inserts a blank here)
- E -- 'ARRAY' declaration
- G -- procedure containing parameters
- H -- procedure without parameters
- K -- semicolon signifying the end of a procedure declaration
- A -- already completely processed
- B -- I/O procedure (inserted by Phase 20)
- blank -- none of the above options

Second character:

- 1 -- converted
- 0 -- not converted

Third character:

- 1 -- semicolon in BUFFER
- 0 -- no semicolon

ELEM1 -- contains a line of 100 characters

THE DICTIONARY

The dictionary is in three parts: DICT, DICEXT, and DICPRO. Different phases of the LCP use these parts in different ways.

- Phase 10 uses DICEXT and DICPRO.
- Phase 20, because of a temporary limitation in PL/I, cannot transfer information directly from SYSUT2 to DICT. It therefore uses DICEXT and DICPRO as buffers between SYSUT2 and DICT. The parts of DICT corresponding to DICEXT and DICPRO are also called DICEXT and DICPRO, and are organized in the same way as the buffers.
- Phase 25 uses only DICPRO.
- Phase 30 uses only DICPRO.
- Phase 40 uses only DICPRO.

DICEXT holds information concerning the identifiers of a block. DICPRO holds information concerning the parameters of a procedure. For example, in the piece of ALGOL program which follows, information relating to the contents of the box is held in DICPRO and information relating to the rest of the program is held in DICEXT.

```
'BEGIN'
   'REAL' X,Y;
   'BOOLEAN''PROCEDURE' EVEN[(I);
       ['VALUE' I;   'INTEGER' I;]
       <procedure body>
       <statement>
            •
            •
            •
       <statement>
'END'
```

DICPRO and DICEXT are described in detail in the following paragraphs.

DICEXT

This section of the dictionary is used for stocking all information concerning the identifiers of a block.

During Phase 10 of the program, DICEXT is placed on SYSUT2 whenever (a) the end of a block has been encountered, or (b) it is required again because a new block occurs before the preceding one has been completed. An incomplete block can be recalled for completion should Phase 10 contain further declarations concerning it.

During Phase 20, DICEXT is stored and serves as a dictionary of active identifiers in a given part of the program. Each division of DICEXT contains the internal reference number of the corresponding block. At each end-of-block, the dictionary section is deleted.

The contents of DICEXT are set out as follows:

IRNEXT	NIDEXT	IDEXT	TYPEXT

IRNEXT -- contains the internal reference number of the line opening the block.

NIDEXT -- contains a count of the identifiers.

IDEXT -- contains a table of up to 172 identifiers.

TYPEXT -- contains a table of 172 x 31 bits, as follows:

	A	B	C	D	E
No. of Bits	11	4	5	1	10

A -- contains a count, incremented by 1, of the dictionary sections containing information about procedures with parameters.

B -- is not used.

C -- contains the number of dimensions of an array or the number of parameters of a procedure.

D -- set to 1 if the converted identifier is to receive the suffix $.

E -- contains declaration information given by a decimal value, as shown in the following table:

Item Declared	Type	Decimal Value
Simple variable	'REAL'	1
	'INTEGER'	2
	'BOOLEAN'	3
Array	'REAL'	5
	'INTEGER'	6
	'BOOLEAN'	7
	Not defined	5
Procedure		
Simple		8
With parameters 'ARRAY' or 'PROCEDURE'		520
In internal code		24
Type Procedure	'REAL'	9
	'INTEGER'	10
	'BOOLEAN'	11
Switch		16

Each ALGOL keyword in the declaration contributes to the decimal value describing the declaration. For example, in the following declaration:

```
'BOOLEAN''PROCEDURE' X; 'CODE';
```

'BOOLEAN' has value 3, 'PROCEDURE' has value 8, and 'CODE' has value 16. The sum, 27, is the decimal value describing a boolean procedure with the body 'CODE'.

If a procedure has parameters specified by 'ARRAY' or 'PROCEDURE', 512 is added to the decimal value of the declaration.

DICPRO

DICPRO stocks all information concerning the parameters of a procedure. It is used in turn by each phase of the LCP.

Phase 10 places DICPRO on SYSUT2 when all parameter specifications have been read and processed.

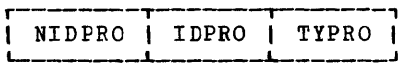
Phase 20 recalls DICPRO when the corresponding procedure declaration has been read. While the procedure is being processed, the descriptions of formal parameters specified by 'ARRAY' or 'PROCEDURE' may be completed; in this case, DICPRO is placed on SYSUT2 at the end of the procedure.

Phase 25 (if called) reads the sections of the dictionary required for processing Table PCT. When the description of a formal parameter has been completed, DICPRO is placed on SYSUT2.

Phase 30 consults descriptions of formal parameters in DICPRO in order to convert parameter specifications.

Phase 40 uses DICPRO to convert actual parameters of procedure (or function) calls and to simulate calls by value.

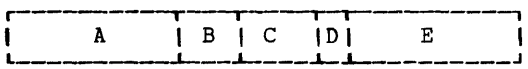
The contents of DICPRO are set out as follows:



NIDPRO -- contains a count of parameters in the procedure.

IDPRO -- contains a table of up to 15 formal parameters.

TYPRO -- contains the description of each formal parameter. It is a table of 15 x 31 bits, as follows:



	A	B	C	D	E
No. of bits	11	4	5	1	10

A -- contains a count, incremented by 1, of the dictionary section containing information about a parameter specified by 'PROCEDURE' or 'ARRAY'.

B -- contains a count of parameters in the dictionary section.

C -- contains the number of dimensions of an array or the number of parameters of a procedure.

D -- set to 1 if the converted identifier is to receive the suffix $.

E -- contains specification information given by a decimal value, as shown in the following table:

Item Specified	Type	Decimal Value
Variable called by name	'REAL'	1
	'INTEGER'	2
	'BOOLEAN'	3
Variable called by value	'REAL'	65
	'INTEGER'	66
	'BOOLEAN'	67
Formal parameter specified by 'ARRAY' and called by name	'REAL'	37
	'INTEGER'	38
	'BOOLEAN'	39
	Not defined	37
Formal parameter specified by 'ARRAY' and called by value	'REAL'	101
	'INTEGER'	102
	'BOOLEAN'	103
	Not defined	101
Formal parameter specified as non-type procedure		40
Formal parameter specified as type procedure called by name	'REAL'	41
	'INTEGER'	42
	'BOOLEAN'	43
Formal parameter specified as type procedure called by value	'REAL'	257
	'INTEGER'	258
	'BOOLEAN'	259
Switch		16
Label		0
String		128

Each ALGOL keyword used as a specifier contributes to the decimal value describing the specification. For example, in the following specification:

'VALUE' A; 'REAL''ARRAY' A;

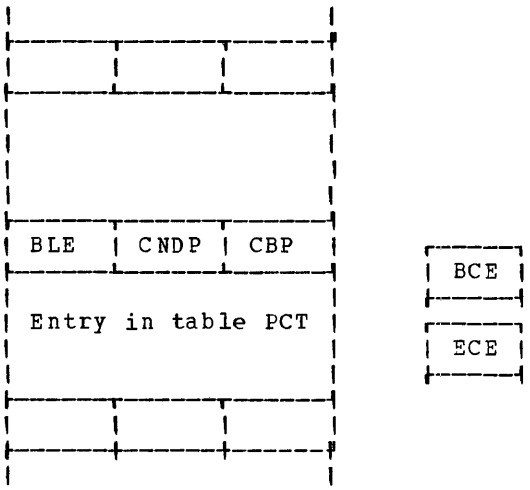
'VALUE' has value 64, 'REAL' has value 1, and 'ARRAY' has value 36. The sum, 101, is the decimal value describing the specification of formal parameter A.

If a parameter is called by value, 64 is added, unless it is a type procedure, in which case 256 is added.

TABLE PCT (PROCEDURE CALL TABLE)

This table contains information concerning procedure calls with actual parameters that are procedures or arrays. It is built in Phase 20 and used in Phase 25 to complete the dicticnary.

When an ALGOL procedure statement is encountered, a stack is used to construct Table PCT. This stack has the following form:



BCE -- Address of beginning of current entry

ECE -- Address of end of current entry

BLE -- Address of beginning of previous entry

CNDP -- Dimension (or parameter) counter

CBP -- Brackets and parentheses counter

Each entry has more than one line. Each line has the following form:

APN	CODE	STN	FPN

CODE -- contains one of the following PL/I symbols:

- PROCM for a procedure
- PROCP for a formal procedure
- ARRAYP for an array or a formal array
- IONAM for an I/O or standard function name

APN -- contains the sequence number of the actual parameter (0 indicates an element creating a new entry).

STN -- contains the section number or the number of parameters or dimensions, depending on the value of CODE.

FPN -- contains the sequence number of the formal parameter when CODE is PROCP or ARRAYP.

The contents of the last entry in the stack are entered in Table PCT. The first line is used to reset the various pointers.

COMAREA

COMAREA is the area of communication among the various phases of the LCP. The layout of COMAREA is as follows:

BLIDIC	BLIPRO	IRN	SIZE	PGPROT	PGPROL

LONG	ISO	CHAR48	SOURCE	DECK	LSTKEY	ERZON

BLIDIC -- contains a count of the blocks in the program.

BLIPRO -- contains the number of procedure declarations (with parameters) in the program.

IRN -- contains the total number of records on SYSUT1.

SIZE -- contains the storage capacity required for DICT in Phase 20. The requirement depends on the SIZE option.

PGPROT -- contains the type of the precompiled procedure (if any).

PGPROL -- contains the procedure identifier if the program is a precompiled procedure.

LONG -- contains the precision option:

1 for LONG

0 for SHORT

ISO -- contains the character-code option:

1 for ISO

0 for EBCDIC

CHAR48 -- contains the character-set option:

1 for 48-character set

0 for 60-character set

SOURCE -- contains the source listing option:

1 if required

0 if not required

DECK -- contains the option of output on SYSPCH:

1 if required

0 if not required

LSTKEY -- contains the key of the record on SYSUT1 containing the last procedure declaration (set by Phase 20 and by Phase 30).

ERZON -- contains the count of syntactically incorrect ALGOL statements.

This appendix contains detailed descriptions of the files used by the LCP.

The file access method for the ENVIRONMENT attribute is given in the PL/I Programmer's Guide, Form C28-6594.

FILE SYSUT1

File SYSUT1 has the following PL/I attributes:

KEYED

RECORD

ENVIRONMENT(REGIONAL(1) F(123))

1. The key used to create and retrieve records is the internal reference number.
2. The record length is fixed at 123 bytes, the same size as BUFFER.
3. The ENVIRONMENT attribute specifies the access method used.

After formatting, file SYSUT1 is used as UPDATE DIRECT.

FILE SYSUT2

File SYSUT2 has the following PL/I attributes:

KEYED

RECORD

ENVIRONMENT(REGIONAL(1) F(1729))

1. For uneven values (1,3,5,...), the key is obtained from the number of the block in DICEXT. For even values (0,2,4,...), the key is obtained from the number of the procedure declaration in DICPRO.
2. The physical records are of fixed length. A logical record corresponding to DICEXT occupies 1429 bytes and a record corresponding to DICPRO occupies 154 bytes.
3. The ENVIRONMENT attribute specifies the access method used.

In the Control Phase, the file SYSUT2 is opened as OUTPUT DIRECT to make the disk ready. It is then closed and re-opened as UPDATE DIRECT.

FILE SYSIN

File SYSIN has the following PL/I attributes:

RECORD

INPUT

ENVIRONMENT(CONSECUTIVE)

FILE SYSPRNT

File SYSPRNT has the following PL/I attributes:

RECORD

OUTPUT

ENVIRONMENT(CONSECUTIVE)

The size of the records is fixed by the DCB subparameter of the DD card. (See the sample program in the ALGOL-to-PL/I language conversion manual.)

The first character of each record is a printer control character.

FILE SYSPCH

File SYSPCH has the following PL/I attributes:

RECORD

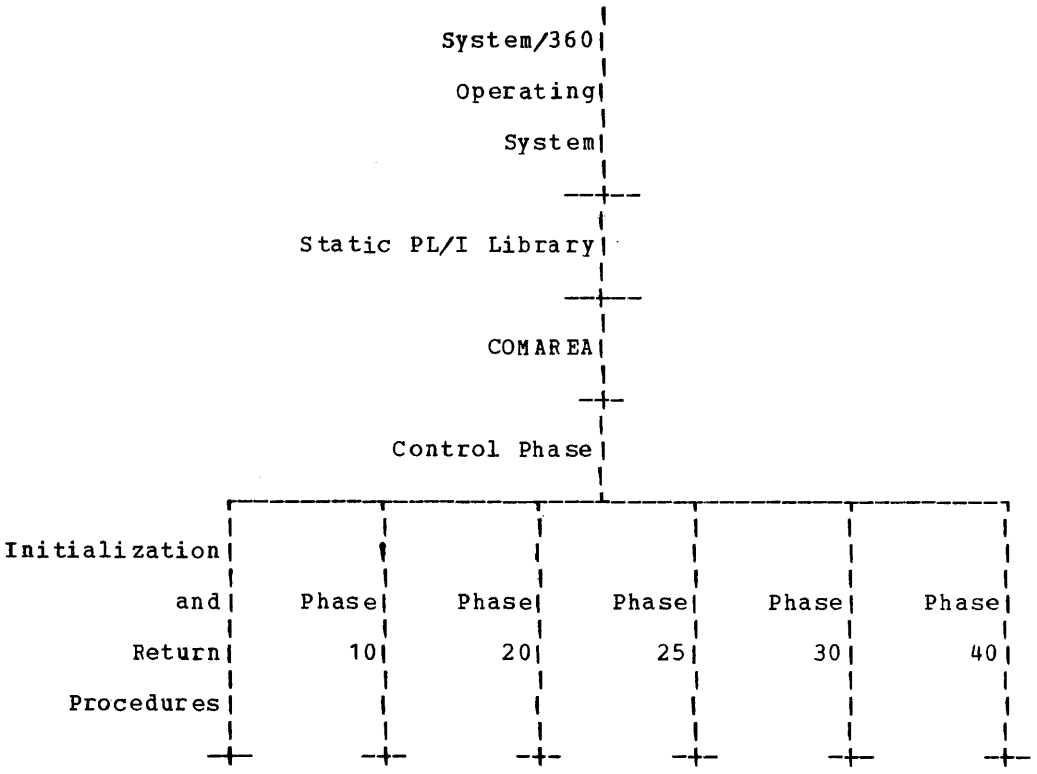
OUTPUT

ENVIRONMENT(CONSECUTIVE)

APPENDIX C: STORAGE MAP

The following diagram illustrates the way in which the LCP uses main storage. The BUFFER, tne Dictionary, and Table PCT are considered as parts of the phases that use them.

The exact sizes of the components of the program are not shown since they depend on the options used to compile the LCP.



Where more than one page reference is given, the first page number indicates the major reference.

READER'S COMMENT FORM

IBM System/360 Conversion Aids:
ALGOL-to-PL/I Language Conversion Program
for IBM System/360 Operating System

Y33-7006-0

- How did you use this publication?

  As a reference source ..... ☐
  As a classroom text ..... ☐
  As a self-study text ..... ☐

- Based on your own experience, rate this publication . . .

	Very Good	Good	Fair	Poor	Very Poor
As a reference source:	.....	.....	.....	.....	.....
As a text:	.....	.....	.....	.....	.....

- What is your occupation? .....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This SRL bulletin is one of a series which serves as reference sources for systems analysts programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Department 813

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Printed in U.S.A. Y33-7006-0

Printed in U.S.A. Y33-7006-0

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]