PLAGO/360 FAST PL/I TRANSLATOR

1.0 Introduction

PLAGO is a fast PL/I translator developed at the Polytechnic Institute of Brooklyn and is designed specifically for use by a large number of students. It features a very fast job time (time needed to translate and run a program), usually well under one second. In addition, a large number of error messages pinpoint mistakes and often help guide the way towards writing a successful program.

1.1 Preparing Input for Plago

All input to PLAGO must consist of 80 column cards punched on an IBM 029 keypunch[1]. These cards are stacked into a card deck which is read by PLAGO in sequence. These cards can be grouped into three categories: 1) Control cards which control the translation and execution of a job; 2) Program cards through which the user instructs the computer about what to do in order to solve his problem; 3) Data cards which contain numbers to be read by the user's program for use in calculations during execution of his program.

It is sometimes necessary to refer to a particular punch or punch combination. Usually, this is most conveniently done by referring to it by row and/or column. As mentioned above, all input cards to PLAGO contain 80 columns. Card manufacturers usually number the columns along the top and bottom of card and the columns can be identified by these numbers. Likewise, rows of numbers ranging from zero to nine are also printed. These rows can also be identified by number. In addition, there are two additional rows above the zero row. The row immediately above the zero row is called the 11 row, with the row above that (the top row on the card) being called the 12 row.

Therefore, since the punch combination for the letter A is a punch in the 12 (top) row and the 1 row in the same column, the punch representation of the letter A is a 12-1 punch. Furthermore, the representation of a comma is a 0-8-3 punch, and so on. Finally, the top and bottom edges of the card are identified by the rows nearest them. Therefore, the top edge of the card is known as the 12 edge and the bottom edge is the 9 edge.

1.3 System Input

Cards are read by PLAGO by means of a high speed card reader operating at about 1000 cards per minute and intermediate storage devices. Information for control cards may be punched between columns 1 and 72. Information on the PL/I language program cards may be punched between columns 2 and 72, with column 1 being ignored. For this reason column 1 is usually left blank and columns 73 through 80 may be used for user identification codes for his own use. Columns 73 through 80 are not read by PLAGO for program cards. Data cards use all 80 card columns and therefore the entire card is scanned for input data.

---

[1] See Appendix A for a keypunch description.

1.4 System Output

All PLAGO output is directed to a high speed printer operating at 450 lines per minute and to intermediate storage devices. The number of printer columns available to the PLAGO user is 120, although the actual number of print positions on the physical printer is 132. The printer is used for source program listings, program execution output, and error messages.

2.0 Control Cards and Job Structure

Running a job on PLAGO requires the use of only three system control cards, which are identified by a dollar sign ($) in column 1 of the card. Every job must utilize all three of the control cards described below. The dollar sign in column 1 is always immediately followed by a control key word. No intervening spaces are allowed. Over each of the following figures of card descriptions, numbers appear stating in which columns the information must start and end.

```
12   6
$BEGIN    acct#name[,options]
```

Fig. 1. Format of a $BEGIN control card

2.1 $BEGIN Card

The first card in the deck is known as a "$BEGIN" card. Its general format is shown in Fig. 1. It must always be the first card in the deck. It has several purposes. First, it notifies the system that the beginning of a new deck is now in the reader and that PLAGO initialization processes should start. Second, it uses the account number to determine whether the owner of the deck is a valid user of the system. If he is, it also determines who is to be billed for his use of the computer. Third, since the programmer's name is on the card it provides a means to sort and match the decks and printouts for eventual return to the user. Fourth, it may be used to specify options to the compiler regarding amount of time needed, number of pages to be printed, and other information, If no options or incomplete options are specified PLAGO assumes default values for the missing options.

Being a control card, it must contain a dollar sign in column 1 and the keyword "BEGIN" in columns 2 through 6, followed by one or more blanks. Next comes the user's 8 character computer account number, followed by one or more blanks. This account number identifies the user to the computer. Next comes the programmer's name, with any character allowed in the name except commas.

The programmer's name may optionally be followed by a comma and a list of options separated by commas. Each option has the following format:

KEYWORD=VALUE

where KEYWORD identifies the option to be utilized and VALUE is a parameter to be assigned to that option. Options available include the following:

TIME=n System will allow your job to execute for n minutes. If this option is omitted, one-half minute is assumed. In general it is a good idea to consult with your instructor or advisor if you need more than the default time, since poor programming technique may be the cause of such a long execution time.

PAGES=n System will allow your job to print up to n pages on the system printer during execution. Any attempt to go beyond this limit will result in an error message and termination of the job. If this option is omitted, 10 pages is assumed. Once again, any decision to increase this limit should be made only after consulting with your instructor or advisor, since you may be producing more output than is necessary.

```
1   5
$DATA
```

Fig. 2. Format of a $DATA Control Card.

2.2 $DATA CARD

The $DATA card serves three purposes: 1) It signals the end of the user's PL/I source programs; 2) It indicates that input data to be read by the programs during execution may be immediately following this card; 3) It initiates the loading and execution of the user's object program. This card must always immediately follow the user's PL/I source programs. Its format is shown in Fig. 2 and has the standard control card format of a dollar sign in column 1 and the keyword DATA in columns 2 through 5. The rest of the card is ignored and should be left blank.

```
1  4
$END
```

Fig. 3. Format of the $END Control Card.

2.3 $END CARD

The $END card indicates the end of the user's input deck to PLAGO. It must always be present and must always be the last card of the user's deck. The $END card format is shown in Fig. 3 and follows the standard control card format of a dollar sign in column 1 and the keyword END in columns 2 through 4. The rest of the card is ignored and should be left blank.

2.4 SUMMARY

This section will summarize the use of control cards in the user's deck:

1. The $BEGIN card issued by the Computer Center is always the first card of the deck.
2. The user's PL/I source programs follow. No control cards are needed between programs.
3. The source programs are followed by a $DATA card.
4. Any data to be read during execution is included next.
5. A $END card is included as the last card in the deck.

A pictorial summary is shown in Fig. 4.

```
$BEGIN    acct.#    name,options
          PL/I Source Programs

$ DATA
          Input Data Cards (if necessary)

$ END
```

Fig. 4. Input Deck Sequence for PLAGO

3.0 PLAGO LANGUAGE SUMMARY

PLAGO is designed for an educational environment with particular emphasis on its use by students of engineering and science. For this reason PLAGO does not support the entire PL/I language but instead a compatible subset of PL/I which is usually used in scientific form. A complete description of the PL/I language as it is implemented on various machines at this time is contained in the IBM PL/I REFERENCE MANUAL, form number C28-8201. A more complete description of the PLAGO language subset is given in Appendix C of this guide. In this appendix is a list of statements implemented in PLAGO and a brief description of the use of each statement. Appendix D of this guide contains a list of differences between the PL/I subset used in PLAGO and the language as described in the IBM PL/I REFERENCE MANUAL.

4.0 DIAGNOSTIC MESSAGES

The present version of PLAGO has only one level of error severity with all errors either preventing execution of the program if it has not started or terminating the job if execution has already been initiated. All errors are noted by means of a code number appearing on the computer printout. A list of error numbers and their explanation as well as descriptions of causes of the errors and possible solutions are given in Appendix B. Errors may be grouped into four categories:

1. Initial translation errors found when PLAGO was first scanning the source program text. The offending column is marked with a dollar sign ($) to indicate exactly where the error was detected. In addition, the error number is printed just to the right of the dollar signs. Errors in this category have number codes between 0 and 99.

2. Program construction errors detected after PLAGO was finished scanning the source text. These errors are indicated by the word "ERROR" followed by an error code and possibly a name indicating a label or variable used incorrectly. All errors of this type are listed immediately following the course listing on the printout and are identified with error codes between 100 and 199.

3. Execution errors caused by a program malfunction. These errors are identified by the keyword "ERROR" followed by the error code and the statement in which the error occurred. All error messages of this type are usually mixed among the program printout and are identified by code numbers between 200 and 299.

4. Errors detected by the PLAGO system in general which can occur during any phase of job processing. Errors of this type cause an English text error message to be printed without any identifying error code number. These messages are always self-explanatory in nature.

APPENDIX A. Use of Keypunches

A1.0 Introduction

The IBM System/360 utilizes a high-speed card reader as its main input device. This reader has two sets of metal brushes internally, with one set of brushes at a slight electric voltage. Normally an electric current is running between the two sets. When the card reader is running, punched cards are sent through a path between the brushes one at a time, insulating the sets of brushes and shutting off the current. Whenever a hole in the card is encountered as the card moves through, current flows between the two sets. In addition, each set of brushes is separated into groups so that the reader will know exactly the position of the hole on the card. By noting the pattern of holes in a card column the reader determines the character read, converts the character into a binary form acceptable to the computer, and sends the encoded character into the computer memory.

Obviously, the task of hand punching a card with the correct hole patterns to indicate the desired characters is a laborious task. A device has been invented which is operated in a manner similar to an electric typewriter, called a keypunch.

A1.1 Operating the Keypunch

There are many different keypunches on the market today which are compatible with the System/360. Of these, the two most popular keypunches are the IBM 026 and IBM 029. Although the characters and hole combinations they punch are somewhat different, their basic use and mode of operation is the same.

The first feature of the keypunch is the path of cards through it. At the top right corner of the keypunch is a card hopper which contains the user's blank cards which are to be punched. The cards are held upright in the hopper by a spring loaded back plate which in turn is held by a lever to keep it from slipping. To put cards in the hopper, reach over the keypunch and locate the back plate and its associated lever. Release the plate by closing the lever against the plate. Then, still holding the lever, push the plate towards the back of the machine until a gap is created which is large enough to hold the cards. Insert the cards and let go of the plate, allowing it to spring back against the cards and hold them in place. At this point the cards are loaded and ready to punch, one at a time.

The remainder of this section can best be read while being seated in front of a keypunch. If this is not possible, then refer to Figs. A.1 and A.2 for diagrams of the path that cards follow when going through the keypunch and the keyboard.

USE OF KEYPUNCHES

Within the keypunch, a card can be in any one of five locations:
1. In the card hopper awaiting processing as described above. This is indicated by a "1" on Fig. A.1.
2. In standby position waiting to be registered in the punch station. This is indicated by a "2" on Fig. A.1.
3. Registered in the punch station ready to be punched or being punched. This is indicated by a "3" on Fig. A.1.
4. Registered in the read station ready to be read or being read. This is indicated by a "4" on Fig. A.1.
5. In the output card stack, with all processing finished for that card. This is indicated by a "5" on Fig. A.1.

The movement of cards through the keypumch is controlled by three buttons located on the right side of the keyboard (see Fig. A.2): the REL (RELEASE) button, the FEED button, and the REG (REGISTER) button. At this point we will trace the movement of a single card through the keypunch.

The first objective is to move a card from the card hopper to the punch station to ready it for punching. This is accomplished by first pushing the FEED button to feed a card from the card hopper to the standby position, and then the REGISTER button to register the card in the punch station. The card is now ready for punching.

The actual method of punching the card is very similar to using a typewriter. The keypunch has two sets of characters for each key: ALPHABETIC shift, corresponding to the lower case (small letters) on a typewriter, and NUMERIC shift, corresponding to the upper case (capital letters). Normally, the keypunch is in ALPHABETIC shift, and can be changed by depressing the NUM (NUMERIC) key at the bottom left side of the keyboard. For example, if the key next to the SKIP key (right side, middle row) is pressed with the keypunch in ALPHABETIC shift, the letter L will be punched. If the keypunch is shifted to NUMERIC the numeral 6 will be punched. If the fourth key on the top row is pressed with the keypunch in ALPHABETIC shift the character STAR (*) will be punched. If the keypunch is put in NUMERIC shift the character DOLLAR SIGN ($) will be punched, The same is true for all the grey keys on the keyboard. If an illegal symbol is attempted to be punched, such as the numeric character corresponding to the letter A, the keyboard will lock due to the error and may be unlocked by pressing the ERROR RESET key if it is available. Otherwise, just tap the BACKSPACE key in the middle of the keypunch. The BACKSPACE key, if held down, will also back up the card to repunch columns. The card will be backed up as long as the BACKSPACE key is depressed until column 1 is reached. It should be noted that once a column on a card is punched, it is punched irrevocably forever. There is no such thing as erasing a punched column.

When the card is finished being punched it must be released from the punch station so that another card can be brought in for punching. This is done by pressing the REL (RELEASE) key. Above the keyboard is a set of switches, one of which is marked AUTO FEED. If the AUTO FEED switch is turned on, a feed sequence as well as a release sequence takes place each time the RELEASE key is pressed.

In summary, each key will perform the following operations:

1. REGISTER key: Will take cards from the standby position and register them in the punch station. In addition, it will take cards just beyond the punch station through a release sequence and register it in the read station. Finally, it will take cards just beyond the read station and put them at the bottom of the output card stack.

2. FEED key: Performs all the operations of the REGISTER key and in addition will move a card from the card hopper to the standby position.

3. RELEASE key:
   a. With AUTO FEED off: Will release cards from the punch and read stations.
   b. With AUTO FEED on: Will release cards from the punch and read stations and also will perform all the operations of the FEED and REGISTER keys.

One final key of interest on the keyboard is the DUP(DUPLICATE) key. With cards in both the read and punch stations, the card in the read station will be duplicated on the corresponding columns of the card in the punch station, column by column, as long as the DUP key is depressed or until the end of the card is reached. If no card is present in the read station, the card in the punch station will simply be spaced over as long as the DUP key is depressed.

ADDITIONAL FEATURES

Above the punch station there is a small window through whi h can be seen a drum. At the the bottom of the drum is a row of numbers and a pointer. The number pointed to corresponds to the column number ready to be punched in the punch station.

Above the keyboard is a switch labelled PRINT. If the switch is ON, the character punched through the use of the keyboard will also be printed at the top of the card above its respective column. If the switch is OFF, no such printing will be done.

On some keypunches, a spring switch labelled CLEAR is present. By flipping the switch momentarily to the ON position and releasing, the entire keypunch can be cleared of cards from all stations, with all cards ending up in either the card hopper or output card stack.

Other switches, levers, and keys are also on the keypunch. These are normally used only by trained professional keypunch operators or in very special applications and as such are beyond the scope of this section.

OPERATING HINTS

In general, if a large number of cards needs to be punched in consecutive order (such as the initial preparation of a program deck) it is a good idea to have the AUTO FEED turned on. If only a few cards need to be punched in non-consecutive order (such as corrections to a deck) it is a good idea to turn off AUTO FEED.

If the keypunch malfunctions, DO NOT ATTEMPT TO FIX THE KEYPUNCH YOURSELF! Report the difficulty to the proper authorities in the Computer Center. They have experience in dealing with such disasters and also have special tools to fix the keypunch quickly. However, many times in the past, relatively easy

repairs have turned into nightmares resulting in calls to IBM for drastic service. Once again, since it cannot be stated sufficiently strongly:

DO NOT ATTEMPT TO FIX THE KEYPUNCH YOURSELF!!!

APPENDIX B.

PLAGO ERROR MESSAGES

Number	Explanation and Typical Causes
0	An identifier is used locally for more than one purpose. For example, the same identifier is used for a statement-label and variable-identifier. Remember, an identifier must have one and only one meaning within a block. That is, as a statement-label, procedure-identifier, parameter, variable identifier or built-in function.
1	Statement cannot be classified. The statement being scanned is not a valid PLAGO statement. The translator skips to the next semi-colon and translation continues.
2	Statement translation incomplete. According to the syntactical rules for statement construction, the statement should be terminated by a semicolon. However, a semicolon was not found in this point in the scan. The translator skips to the next semicolon and translation continues. This error also occurs when another
3	External procedure has not been recognized. The first statement in the program was not a valid PROCEDURE statement. The translation is terminated.
4	THEN missing in an IF statement. The THEN clause which must follow the expression of the IF statement cannot be found. The translator assumes the existence of the THEN keyword and continues the statement translation.
5	Left parenthesis missing. A left parenthesis is required at this point in the statement. The translator assumes the existence of a left parenthesis and continues the statement translation.
6	Right parenthesis missing. A right parenthesis is required at this point in the program. The translator assumes the existence of a right parenthesis and continues the statement translation.
7.	Invalid variable list in an assignment statement. The variable list in an assignment statement is illegal. The translator skips to the next semicolon and translation continues.
8.	Equal Sign missing in an assignment statement. An equal sign is required at this point in the program. The translator assumes the existence of an equal sign and continues the statement translation.
9.	Invalid LIST in a GET or PUT statement. The LIST or DATA options of a GET or PUT statement are illegal. Remember the DATA option may have no LIST or specify variable identifiers, the LIST for a GET statement must be composed of expressions. The translator skips to the next semicolon and translation continues.
10.	LIST or DATA option missing in a GET statement. One of these options must must be present in a GET statement. The translator skips to the next and translation continues.

Number Explanation and Typical Causes (continued)

11. Invalid printer control options in a PUT statement. An invalid combination of printer control options have been specified. The only valid combination is PAGE and LINE.

12. Expression Operand cannot be recognized. A valid constant or identifier is required at this point in an expression translation. Expression translation stops and additional errors may be listed.

13. Not assigned at this time

14. Control variable not recognized in a DO statement. The control variable cannot be recognized, the translator skips to the next semicolon and treats the DO group as a non-iterative DO group.

15. Invalid GOTO statement. An identifier is not found following the GOTO keyword. The translator skips to the next semicolon and translation continues.

16. Invalid CALL statement. An identfier is not found following the CALL keyword. The translator skips to the next semicolon and translation continues.

17. Invalid parameter list. An element of the parameter list is not an identifier. The translator skips to the next semicolon and translation continues.

18. Procedure identifier list missing. A procedure identifier list must be specified prior to the PROCEDURE statement. The translator ignores the absence of the list and translation continues.

19. Invalid precision. The precision following the FLOAT attribute is either not a decimal integer constant, or it exceeds the maximum of 14 digits. In the former case, float single is assumed. In the latter case, float double is assumed. In both cases, the statement translation continues.

20. Invalid DECLARE statement. An identifier is executed at this point in the statement. The translator skips to the next semicolon and translation continues.

21. The declaration of a parameter using the asterisk notation requires that all dimensions be specified and/or all lengths be specified with asterisks. The translator skips to the next semicolon and translation continues.

All of the above errors will be printed below the erroneous source program statement, with a dollar sign ($) pointing to the column in question.

SECOND PASS ERRORS

All these errors appear at the end of the source listing. They are noted only after PLAGO has already scanned the source program one time.

100. Identifier in a GOTO statement is not a statement label. The identifier in a GOTO statement, is not known globally or locally as a statement label.

SECOND PASS ERRORS (cont'd.)

Number Explanation and Typical Causes

101. Illegal transfer of control into a DO group or block. Control may not be transferred into a DO group, procedure block, or BEGIN-END block via a GOTO statement. Remember, the DO statement of a DO group, or the BEGIN statement of a BEGIN-END block must be executed by a flow of control; a procedure block must be entered by a CALL statement or procedure function invocation. The illegal statement label follows the statement number.

102. Statement label illegally used. An operand in an expression, or a variable in a variable list specifies a statement label. The illegal statement label follows the statement number.

103. Illegal function or subscripted variable reference. A reference was made in the program which is not a valid subscripted reference to an array, not a valid procedure function reference. The illegal identifier follows the statement-number. Usually caused by failing to include the variable in a DECLARE statement or the DECLARE statement in which the variable appears is syntactically incorrect.

104. Incorrect number of arguments or subscripts. A subscripted reference was made to an array and the number of subscripts does not match the number of dimensions declared for the array. Or a reference was made to a built-in function where the number of arguments does not match the number of required arguments for the built-in function. Or a procedure function reference or CALL statement reference was made where the number of arguments does not match the number of parameters declared for the procedure. The identifier of the illegal reference follows the statement number.

EXECUTION ERRORS

These errors occur during execution of the program and are mixed among the values put out by the user's program. The format of the message is as follows:

ERROR error number IN STATEMENT statement number

200. An array expression has been used in a context where a scalar expression must appear.

201. Declared character string length exceeds the maximum of 255 characters.

202. The memory space allocated for data storage has been exhausted.

203. An attempt is being made to assign a value to a parameter where the corresponding argument is not a legal target. For instance, a constant or temporary.

204. Lower bound of an array dimension is not less than the upper bound.

EXECUTION ERRORS (cont'd.)

Number	Explanation and Typical Causes
205	X = 0 and Y < 0 in X ** Y
206	Arrays whose bounds are not identical have been used in an array expression.
207	X < 0 in SQRT(X)
208	X ≤ in LOG(X) or LOG2(X) or LOG 10(X)
209	ABS(X) ≥ K * 2^{50} in SIN(X) or COS(X) (K=π) or in SIND(X) or COSD(X) (K=180)
210	X > 174.673 in EXP(X)
211	ABS(X) ≥ K * 2^{50} in TAN(X) (K=π) in TAND(X) (K=180)
212	X too close to singularity in TAN(X) or TAND(X)
213	Y = 0 in MOD(X,Y)
900 901	Error in execution interpreter

ON CONDITIONS

ENDFILE	Attempt to read more data then was supplied
NAME	For GET DATA, the name of the data item listed was not specified on the list (or is not known if no list was specified). The item is skipped and execution continues.
FIXEDOVERFLOW	The result of a fixed binary operation exceeds $2^{31} - 1$.
OVERFLOW	The exponent of a floating point number exceeds 75.
UNDERFLOW	The exponent of a floating point number is less than -79. Execution continues.
CONVERSION	An illegal conversion from character string to numeric value has been attempted. This condition is also raised for invalid input data.
SUBSCRIPTRANGE	The value of a subscript lies outside the specified bounds.
ZERODIVIDE	Division by zero in either a fixed binary or floating point operation.
STRINGRANGE	The parameters to the SUBSTR function are such that the specified substring lies outside the string. Execution continues using the valid part of the substring, which may be null. If both sides of a statement using SUBSTR are invalid, this condition is raised twice.

ON CONDITIONS

ERROR	This condition is raised following any on-condition which terminates execution.
FINISH	This condition is raised immediately before the job is terminated.