APPENDIX C

PLAGO LANGUAGE SUMMARY

The following pages are a guide to the PL/I language features which are implemented in PLAGO Version I. This summary is not a tutorial treatment of how to use the language, nor is it a rigorous definition of the language facilities. The summary is simply reference material which will be valuable to the student in constructing PLAGO programs as he masters the features of PL/I.

The major section headings of the summary are as follows:

LANGUAGE FEATURES	COMMENTS
BASIC SYMBOLS	
character set the following characters are available: 26 upper case letters, 10 decimal digits, plus the special characters, = + - * / ( ) , . ' % ; : ¬ & | > < _ ? □ $ @ #	The EBCDIC character set is utilized. □ indicates a blank character.
BASIC SYNTACTICAL CONSTRUCTIONS	
identifier A letter followed by a maximum of 30 letters, digits or the underscore _. ex: A RATE NET_INCOME	The first 8 characters must be unique in order to distinguish between different identifier usages. Used for variables, statement-labels, procedure-identifiers, built-in function identifiers, and pseudo-variable identifiers.
constant Fixed-point decimal, floating-point decimal and character string constants are permitted. ex: 5 5.6 3.14159 65E-2 753.82165E3 'ABC' '' 'IT''S'	1. Decimal integer constants with a value $< 2^{31}-1$ become fixed-point binary data of precision 31, otherwise, they become float double if $\geq 2^{31}-1$. 2. Non-integer decimal constants always become float-single or float-double based upon the value magnitude. 3. Floating-point constants become float-single if mantissa is $< 2^{24}-1$, otherwise float-double. All floating-point values are hexadecimal. 4. Character string constants may be null or contain a maximum of 255 characters. 5. Fixed-point and floating-point constants may not contain more than 14 significant digits.

LANGUAGE FEATURES	COMMENTS
expressions All of the PL/I operators may be used in constructing expressions. Parenthesized subexpressions may be utilized. (See examples following operators)	Conversions are performed when operand types differ. Expressions may be array or scalar expressions. If more than one array is utilized, all arrays must have identical bounds. The contexts in which a scalar expression is required are denoted below.
operators (by precedence levels) prefix +, prefix -, ¬, ** *, / infix +, infix - || =, >, <, >=, <=, ¬=, ¬>, ¬< & | ex: A 5 .65E-2 A+B A-B*C (A+B)/(C+D) A>B | C<D X || Y A+B & C-D	 Integer division produces an integer result (fraction cat Fixed-point binary 1 if true, 0 if false. Fixed-point 1 if both operands are non-zero, else 0. Fixed-point 1 if either operand is non-zero, else 0.
subscript reference An array variable identifier followed by a subscript list in parenthesis. identifier (expression-list) ex: A(3) MATRIX(I,J+K)	The subscripts must be scalar expressions. The number of subscripts must be equal to the number of dimensions declared for the array. Note: cross section array references are not permitted.
built-in function A built-in function identifier, followed by an argument-list in parenthesis. identifier (expression-list) ex: ABS(X) SIN(T+G)	The arguments in most cases may be any expressions, the of arguments must be equal to the number of arguments required by the built-in function. (See G for a list of the built-in functions)

LANGUAGE FEATURES	COMMENTS
identifier-list One or more array or scalar variable identifiers separated by commas. identifier$_1$, identifier$_2$, ..., identifier$_n$ ex: ALPHA, X, BETA, Y	Used in data-directed GET and PUT statements. Also used for specifying procedure parameters.
variable-list The same as an identifier-list, except that subscripted references may be used. ex: X(I), Y, M(3,3)	Used as the target list in assignment statements and list-directed GET statements.
expression-list One or more array or scalar expressions separated by commas expression$_1$, expression$_2$, ..., expression$_n$ ex: A+B, C(I)-D, V, E>X	Used as subscript references, argument lists and list-directed PUT statements.
pseudo variable Used to treat a part of a variable or two variables as a target. The only pseudo variable available in this version is SUBSTR (i.e. substring). SUBSTR (e_1, e_2, e_3) e_i = expression	SUBSTR defines a substring to be used as a target, it may be used anywhere the regular variable is used for a target. e_1 specifies the source string. e_2 specifies the beginning of the substring. e_3 if present, specifies the length of the substring. if absent, the length is from e_2 to the end of the e_1.
comments Used to place documentation in the text of the program as follows: /* any characters */	Comments may be placed before and after any statement in the source program.

LANGUAGE FEATURES	COMMENTS
Note: One or more identifiers may be placed prior to any statement in the program. Each identifier must be followed by a colon (:). If the statement is a PROCEDURE statement, then the identifiers are procedure-identifiers. In all other cases, the identifiers are statement-labels.	
C. BASIC STATEMENTS	

<u>assignment statement</u> assigns the result of an expression to one or more targets in a variable list.
 	
`variable-list = expression;`
`ex: A = 3;  V = X+Y;  A,B = D*E;`
`    M(I,J), X, Y = X/Y;`
`    COMPUTE:  VALUE = (X+Y)/Z;`
Note the statement label.
	If the expression result is scalar, then the variables in the variable-list may be scalar or array variables. If the expression result is an array, then the variables in the variable-list <u>must</u> also be arrays with identical bounds.

<u>IF statement</u> selects the next statement to be executed based upon an expression result.
 	
IF expression THEN statement$_1$;
  ELSE statement$_2$;
`ex: IF A<B THEN A = B+C;`
`    IF X THEN GOTO NON_ZERO;`
`    IF X>Y| X<Z THEN X = 0;`
`       ELSE X = Y*Z;`
	The expression <u>must</u> be a scalar expression. If the expression result is true (non-zero), then statement$_1$ is executed. If the result is false (zero), then statem is skipped. The ELSE clause is optional. If present an the result is true, then statement$_2$ is skipped after the execution of statement$_1$. If false, statement$_1$ is skipped and statement$_2$ is executed.

<u>GOTO statement</u> Transfers program control to the next executable statement following the statement-label.
GOTO identifier; or GO□TO identifier;
`ex: GOTO DONE;`
`    GO□TO COMPUTE;`
	The identifier must be a statement-label which is known at that point at which the GOTO statement is executed.

LANGUAGE FEATURES	COMMENTS
STOP statement Causes the program to be terminated. STOP;	The execution of this statement causes the signalling of the FINISH condition.

GET statement Causes the transmission of data from SYSIN and assigns values to target variables. There are two forms of the GET statement.
 	
list-directed
 	
GET LIST (variable-list);
 	
data-directed
 	
GET DATA (identifier-list);
EXAMPLES:
GET LIST (X,Y,Z);
GET LIST (A(I), B(J));
GET DATA (A,B,C);
	An ENDFILE condition will be signalled during the executio of the statement if this condition arises. In list-directed, if an array variable is specified, the next n values, based upon the dimensions, are acquired in row major order. If the first data-directed format is specified, then only the variables specified become eligible for assignment.

PUT statement Causes expression values or variables to be displayed on SYSPRINT. There are two forms of the PUT statement. list-directed PUT LIST (expression-list); data-directed PUT DATA (identifier-list);	The standard tab positions are used with five tabs, where each zone has 24 characters. In list-directed, if an array expression is specified, then n values, based upon the dimensions, are listed in row major order. In the first data-directed format, if an array variable identifier is specified, all elements of the array are displayed in row major order.

LANGUAGE FEATURES	COMMENTS

PUT statement (continued)
 	
The following <u>printer control options</u> may be specified in any order following the PUT keyword. The LIST or DATA options may or may not be present in this case.
 	
PAGE
LINE (expression)
SKIP (expression)
PAGE LINE (expression)
 	
ex: PUT LIST (A,B,C);
PUT LIST (X(I)+Y(I));
PUT DATA (X,Y,Z);
 	
PUT PAGE;
PUT LIST (A,B) PAGE LINE (3);
PUT SKIP DATA (A,B);
	If any of the printer control options are specified, and a LIST or DATA option is specified, the printer control options are executed first. The expressions in the LINE and SKIP options <u>must</u> be scalar expression If the (expression) is omitted from the SKIP option, SKIP (1) is assumed.* Only the PAGE and LINE may appear together in the same statement.
	
	*IF THE EXPRESSION IS ZERO OR NEGATIVE, OUTPUT ITEMS WILL BE OVERPRINTED ON THE SAME LINE

<u>null statement</u> No operations performed upon execution.	Primarily used to resolve dangling ELSE clauses in nested IF statements.
D. DO GROUPS	

<u>statement grouping</u> Provides the means for treating a group of statements which are to be executed once as a single statement.
 	
DO;
statements
END;
	Statement grouping is particularly useful in constructi the THEN and ELSE clauses of an IF statement. For example:
	
	IF expression THEN DO; .... END;
	ELSE DO; .... END;

LANGUAGE FEATURES	COMMENTS

WHILE DO group Provides for the iterative execution of a group of statements while an expression value is true (i.e. nonzero). When the expression is false (i.e. zero), the DO group is terminated.
 	
DO WHILE (expression);
statements
END;
ex: DO WHILE (X< Y);
    S = X*C; X = X+C;
   END;
	The expression must be a scalar expression. This expression is evaluated before the first execution of the DO group, and prior to each successive execution.

specification DO group Provides for assigning values to a control variable based upon specifications and iteratively executing a group of statements until the specification limits have been met. DO variable = specification$_1$, specification$_2$, ..., specification$_n$; statements END; A specification may be one of the following. a. expression b. expression$_1$ TO expression$_2$ BY expression$_3$ c. expression$_1$ BY expression$_3$ TO expression$_2$ d. expression$_1$ TO expression$_2$ e. any of the previous specifications followed by WHILE (expression).	The control variable must be a scalar variable or sca element of an array. All expressions used in the specifications must be scalar expressions. The specifications are used from left-to-right. The DO group terminated when all of the specifications have been utilized. The following describes the actions perfor for each possible specification. a. The expression value is assigned to the control variable, and the DO group is executed once. b. The expression$_1$ value is assigned to the control variable. The values of expression$_2$ and expression$_3$, assigned to two temporary variables E2 and E3. Prior and following each execution of the DO group, the following expression is evaluated. $(E3 >= 0) \& (V > E2) \mid (E3 < 0) \& (V < E2)$ If this expression is true, the specification is termi otherwise, the DO group is executed.

LANGUAGE FEATURES

specification DO group (continued)

```
ex:  DO I = J;
       :
      END;
      DO K = 1 TO 5;
       :
      END;
      DO J = I+2 TO L BY N;
       :
      END;
      DO A(I) = X+Y WHILE (A(I+1));
       :
      END;
      DO K = L1 TO M1, L2 TO M2;
       :
      END;
```

COMMENTS

c. Same as (b). TO and BY clauses may be interchanged.

d. Same as (b). The constant value 1 is substituted for E3.

e. The expression result must also be true, when used with (b, c, or d) or must be true with (a) in order to execute the DO group. If the expression value becomes false, the specification is terminated.

Note: The expression in the WHILE clause is reevaluated prior to each iteration, whereas the other specification expressions are evaluated once and cannot be changed during the execution of the DO group. The last value assigned to the control variable is in the control variable upon termination of the DO group.

Note: DO groups may be nested, however, they must be perfectly nested where each DO statement is terminated by a corresponding END statement. Control must not be transferred to a statement within a WHILE or specification DO group. Entrance to the DO group must always start at the DO statement. If a transfer of control is made to a statement outside of the DO group, the DO group is terminated and can only be reactivated by reentering at the DO statement. If the DO group is reentered in a specification DO group the specifications are used again from left-to-right.

LANGUAGE FEATURES	COMMENTS
DECLARATIONS	
DECLARE statement Specifies the explicit data attributes of variables which are local to the block in which the statement appears. DECLARE identifier_1 attributes_1, identifier_2 attributes_2, ... identifier_n attributes_n; DECLARE may be abbreviated to DCL.	The declaration of the identifier always makes the identifier local to the block in which it is declared. If data attributes are not specified, then these attributes are supplied by implicit declaration. The bounds of an array and/or length of character variables may be scalar expressions when declared in an internal procedure or begin-end block. The bounds and/or lengths may also be specified by asterisks (*) for parameters. (See parameter discussions)
If an array variable is declared, the bounds of the array must be specified in parenthesis immediately following the identifier. A bound is specified as follows: lower-bound: upper-bound where: lower-bound $<$ upper-bound If the lower-bound is omitted, 1 is assumed. The number of dimensions is equal to the number of bounds. $(\text{bound}_1, \text{bound}_2, \ldots, \text{bound}_n)$ The specification of attributes is optional. The permissible attributes are as follows: a. FIXED b. FLOAT (precision) c. CHARACTER (length) d. CHAR (length) e. (c or d) followed by VARYING or VAR	The following describes the meaning of each attribute and its System/360 data representation format. a. Fixed-point binary integer. Four bytes. Value range $\pm 2^{31}-1$. b. All floating-point values are hexadecimal. Precision must be a decimal integer constant of value ≤ 14. If precision ≤ 6 then a single float value is allocated. Four bytes. If precision $> 6 \leq 14$ then a float double value is allocated. Eight bytes. Value range for both precisions is approximately $\pm 5.4E-79$ to $\pm 7.2E75$. If (precision) is omitted, single float is assumed by default. c. and d. The length must be in the range 1 to 255. The length determines the number of bytes. References to fixed length strings uses the entire string. e. The length specifies as in c or d is the maximum length. References to varying strings use the current length.

LANGUAGE FEATURES	COMMENTS

DECLARE statement (continued)
 	
ex: DECLARE A FIXED, B FLOAT (6);
DCL X(10), Y(-10: 10);
DCL RATES (I,J) FIXED;
DCL TEXT CHARACTER (80);
DCL MESSAGE CHAR (50) VAR;	

implicit declaration For variables that are not locally declared via a DECLARE statement, but simply referenced in a block, the following occurs. If the identifier is explicitly declared in a containing block, then the identifier is the same as the explicitly declared identifier. Otherwise, it is moved to the external procedure and acquires its attributes as follows. If the first letter of the identifier is I, J, K, L, M or N, then it is FIXED. Otherwise, it is FLOAT (6). This is referred to elsewhere as the (I-N) rule.	It is important to note that variables may only be explicitly declared via a DECLARE statement. However, statement-labels and procedure-identifiers are explicitly declared by their position in the program, and are local to the block in which they appear. Consequently, they are involved in resolving an implicit declaration.
F. BLOCKS AND BLOCK FEATURES	
procedure-block Provides for identifying the beginning and the ending of an external procedure block.	A program is composed of a single external procedure. A PROCEDURE statement must be the first statement in the program, and an END statement must be the last statement.

LANGUAGE FEATURES	COMMENTS
STORAGE ALLOCATION AND BLOCK TERMINATION Upon entrance to a procedure block, storage is dynamically allocated for all variables which are local to the block. If array and/or string variable bounds and lengths are specified with expressions other than decimal integer constants, then the amount of storage required for these variables is calculated upon block entrance. However, in this case the values of any variables used in the expressions, must be global data variables allocated by an outer block. Upon exit from the block, all storage allocated upon entrance to the block is deallocated. A normal exit from a procedure block is by encountering the END statement of the block.	
multiple closure Provides for terminating DO groups and/or blocks at the same point in the source program without specifying an END statement for each DO group or block to be terminated. ex: LOOP: DO I = 1 TO 10; DO J = 1 TO 10; . . . END LOOP By specifying the statement-label Loop after the END statement, the affect is the same as if two END statements were specified without the statement-label. A: B PROC; ⋮ BEGIN ⋮ DO WHILE (X > Y); ⋮ END B;	The identifier specified following the END statement may be a statement-label or procedure-identifier. All open DO groups or blocks are closed up to and including the DO group or block in which a match is found in a procedure-identifier list prior to a PROCEDURE statement or in a statement-label list prior to a DO statement.

LANGUAGE FEATURES	COMMENTS
multiple closure (continued) In this case, the affect is the same as if three END statements were specified without the procedure-identiifier.	
G. BUILT-IN FUNCTIONS	

In the following list of built-in functions provided by PLAGO, we shall use the following notation to denote the type of arguments that are required in referencing each built-in function:

e - scalar or array expression	se - scalar expression
ae - array expression	ai - array variable identifier

Where arguments to a built-in function are optional, they are underlined. A detailed discussion of the built-in function is contained in the textbook. However, you should note that PLAGO is more permissive, since it allows scalar expressions which are converted to integers, where decimal integer constants are required in PL/I.

ARITHMETIC

ABS(e)
CEIL(e)
FLOOR(e)
MAX(e_1, e_2, ..., e_n)
MIN(e_1, e_2, ..., e_n)
MOD(e, e)
SIGN(e)
TRUNC(e)

MATHEMATICAL

ATAN(e, e)
ATAND(e, e)
COS(e)
COSD(e)
EXP(e)
LOG(e)

LOG10(e)
LOG2(e)
SIN(e)
SIND(e)
SQRT(e)
TAN(e)
TAND(e)

LANGUAGE FEATURES	COMMENTS

LENGTH(e)
SUBSTR(e, e, e)
REPEAT(e, se)
	<u>ARRAY DIMENSION AND ARRAY CALCULATIONS</u>
	PROD(ae)
	SUM(ae)

H. MISCELLANEOUS FEATURES	
<u>ON conditions</u> During the execution of an object program, there are several conditions that may arise. When a condition arises, we say that it is signalled. The ON conditions which are enabled during the execution of the object program are as follows: ENDFILE(SYSIN) NAME(SYSIN) FIXEDOVERFLOW UNDERFLOW OVERFLOW ZERODIVIDE CONVERSION SUBSCRIPTRANGE STRINGRANGE ERROR FINISH When a condition is signalled, the signalling is denoted by placing a message line on SYSPRINT in the following format: CONDITION condition SIGNALLED IN STATEMENT statement-number	On statements are not implemented in this version. However, when a condition arises, the <u>standard system</u> action is exercised as follows: ENDFILE(SYSIN) message - signal ERROR NAME(SYSIN) message - ignore erroneous assignment format and execution continues FIXEDOVERFLOW message - signal ERROR OVERFLOW message - signal ERROR UNDERFLOW message - result set to zero and execution continues ZERODIVIDE message-signal ERROR CONVERSION message - signal ERROR SUBSCRIPTRANGE message - signal ERROR

LANGUAGE FEATURES	COMMENTS
where: condition is one of the above conditions. statement number is the statement in which the condition was detected.	STRINGRANGE SUBSTR reference is forced to be in the string and execution continues. ERROR message - signal FINISH FINISH message - program terminates