60388300

CONTROL DATA CORPORATION

# CYBER PL/I
# GENERAL INFORMATION MANUAL

**CDC® OPERATING SYSTEMS:**
**NOS**
**NOS/BE**

| REVISION RECORD | |
|---|---|
| **REVISION** | **DESCRIPTION** |
| A | Original Release |
| (08-01-77) | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| Publication No. 60388300 | |

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:

**CONTROL DATA CORPORATION**
*Publications and Graphics Division*
**215 MOFFETT PARK DRIVE**
**SUNNYVALE, CALIFORNIA 94086**

or use Comment Sheet in the back of this manual

# LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

| Page | Revision |
|---|---|
| Cover | – |
| Title Page | – |
| ii | A |
| iii/iv | A |
| v/vi | A |
| vii | A |
| 1-1 | A |
| 2-1 | A |
| 3-1 | A |
| 4-1 | A |
| 5-1 | A |
| A-1, A-2 | A |
| Cmt Sheet | A |
| Cover | A |

| Page | Revision |
|---|---|

| Page | Revision |
|---|---|

# PREFACE

This general information manual introduces Control Data Corporation's CYBER PL/I programming language. As described in this publication, CYBER PL/I operates under control of the following operating systems:

NOS 2 for the CONTROL DATA® CYBER 170 Models 171, 172, 173, 174, and 175; CYBER 70 Models 71, 72, 73, and 74; and 6000 Series Computer Systems

NOS/BE 2 for the CDC® CYBER 170 Series; CYBER 70 Models 71, 72, 73, and 74; and 6000 Series Computer Systems

This manual is directed to individuals with some knowledge of computer languages. The reader should be familiar with a Control Data operating system and be acquainted with the basic concepts of FORTRAN, COBOL, and CYBER Record Manager. A knowledge of other software products, such as ALGOL, is beneficial but not required.

As the Control Data implementation of CYBER PL/I version 1 is essentially the same as other PL/I compilers in common use today, references to PL/I in this publication are interchangeable with CYBER PL/I. Where the term CYBER PL/I is used, it is intended to point up a difference between the Control Data implementation and the American National Standards Institute (ANSI) standard for PL/I.

# CONTENTS


1. PL/I – A PROGRAMMING LANGUAGE 1-1

2. PROGRAM ORGANIZATION 2-1

Block Structure 2-1
Group Structure 2-1
Nesting 2-1

3. FLOW OF CONTROL 3-1

Block Activation 3-1
Storage Allocation 3-1
Recursion 3-1
Conditions 3-1

4. DATA HANDLING 4-1

Data Types 4-1
Data Aggregates 4-1
Arrays 4-1
Structures 4-1
Data Storage 4-1
Built-in Functions 4-1

5. INPUT/OUTPUT PROCESSING 5-1

Files and Datasets 5-1
Input/Output Transmission 5-1

APPENDIX

A KEYWORDS A-1

# PL/I - A PROGRAMMING LANGUAGE 1

PL/I was developed to fill the growing need for a programming language suitable for both scientific and commercial users. As computers grew in size and speed, established computer programming languages, such as FORTRAN and COBOL, did not evolve to take advantage of many of the new hardware features. The rift between scientific and commercial programming applications was widening while the hardware was becoming more capable of coping with a mixture of commercial, scientific and systems applications.

PL/I was created by selectively incorporating desirable features of established programming languages while avoiding the less desirable facets of those languages. The new language bears a strong resemblance to FORTRAN, COBOL, and ALGOL, easing the transition for experienced programmers to PL/I. Moreover, beginning programmers need only learn those parts of the language that satisfy their needs. The modular design and extensive use of defaults (for values not specified) make it easier for the novice to start writing programs. As familiarity with PL/I grows, the programmer can use more advanced features and take advantage of aspects of the language to program more efficiently than in other high-level programming languages.

As the number of PL/I users grew, various implementations of PL/I appeared, creating a need for standards to be imposed. The European Computer Manufacturers Association (ECMA) and the American National Standards Institute (ANSI) jointly developed a standard for PL/I. The PL/I language standard was approved by ANSI in 1976 and is published as ANSI X3.53-1976 under the title, "Programming Language PL/I". Control Data's CYBER PL/I conforms essentially to this ANSI specification. The few features that have not been implemented in CYBER PL/I version 1 include:

Complex variables

DEFAULT statement

Aggregate expressions

File, format, and entry variables

Data-directed input/output

Generic functions

A significant plus available with CYBER PL/I is the handling of input and output by CYBER Record Manager, which allows the user of CYBER PL/I to use files produced by other programming languages.

The following sections give an overview of the capabilities available to the programmer writing in CYBER PL/I. Readers well acquainted with PL/I can proceed to section 5 and appendix A. The list of CYBER PL/I keywords in appendix A is probably of greatest interest to the experienced PL/I programmer.

The rudimentary elements of PL/I include keywords, names, literal constants, and delimiters. These are quite similar to their counterparts in FORTRAN and COBOL, and are easily learned. When properly combined, the rudimentary elements form statements, expressions, and references. When these are properly organized (blocked), a program exists. There is no requirement in PL/I that the entire language be mastered before it can be used. A basic understanding of each of the above facets will allow the novice programmer to write programs that will run successfully. Numerous defaults and automatic assumptions aid the novice, often without requiring an awareness of their existence. In fact, many users, more interested in obtaining solutions to problems than in programming skill, seldom venture much deeper into PL/I.

## BLOCK STRUCTURE

The high-level structural components of PL/I are groups of statements called blocks. A block is a delimited area of a program containing some combination of statements and other blocks. A PL/I program consists of one or more blocks. Every statement in a PL/I program is part of a block.

The two types of blocks are begin blocks and procedure blocks. Procedure blocks are often referred to as procedures. A begin block is recognizable by the fact that it starts with a BEGIN statement and terminates with an END statement. A procedure block starts with a PROCEDURE statement and terminates with an END statement.

The major difference between a begin block and a procedure block is the manner in which each is executed. Execution flows sequentially into begin blocks, statement by statement. Procedure blocks are executed only when they are invoked. Each procedure can be thought of as a subroutine. Procedures can invoke other procedures, and these procedures or subroutines can be compiled separately, or they can be nested in a calling procedure and compiled with it. Each procedure can contain declarations that define names and control allocation of storage.

In addition to influencing the flow of a program, blocks delimit areas within a program in which declared names are known. A name declared in one block, for example, might not be known in another block simply because of the relationship of the two blocks to each other. This phenomenon, called the scope of a name, is determined by the position at which the name is declared within the program. A name is known only within the block in which it is declared and in blocks nested within that block.

Other high-level programming languages have lists of keywords that are reserved to the compiler and cannot be used by the programmer. Keywords are not reserved in PL/I because they are recognized by the context in which they appear. The same names can be declared in other blocks with other values and will not cause confusion so long as the rules of blocking are observed.

## GROUP STRUCTURE

A group is a sequence of statements within a block that starts with a DO statement and ends with a corresponding END statement. Thus, a group is also called a DO group, and functions much like FORTRAN DO loops. In PL/I, the DO group allows three different forms of the DO statement: the simple DO, the DO-WHILE, and the indexed DO.

## NESTING

Any block can contain one or more other blocks. A procedure block can be either an internal or external procedure. An external procedure is a procedure block not contained in any other block. The remaining procedure blocks are called internal procedures because they are contained in other blocks.

Blocks contained in other blocks are also called nested blocks, and these can also have blocks nested within them. The outermost block must be a procedure.

The manner in which blocks are nested controls the area of the program in which the scope of a declared name is known; influences the allocation and freeing of certain kinds of data storage; and exerts an influence on the flow of control through the program.

The ordering of statements within blocks and the nesting of blocks within other blocks controls the manner in which the program is executed. Control of the program is determined by the order in which the statements are executed. Flow of control begins with the activation of the main procedure block in a PL/I program. Control passes sequentially from statement to statement in a block until one of the statements causes an alternate path to be taken.

The statements that can change the flow of control are:

CALL

END

RETURN

GOTO

IF

DO

## BLOCK ACTIVATION

When the first or main procedure is activated, the first statement in the program is executed and control starts to pass from one statement to the next. When control passes through the BEGIN statement for a block in the main procedure, that begin block is said to be activated. Activated blocks remain active until they are terminated.

Procedure blocks are activated by a CALL statement or function reference and remain active until an END or RETURN statement terminates them. At that point, control returns to the statement following the CALL statement.

## STORAGE ALLOCATION

PL/I provides four classes of storage allocation, thus allowing the programmer to choose the desired degree of control. Storage allocation means associating an area of storage with a variable.

All variables declared STATIC are allocated storage before program execution begins, and this allocation remains unchanged for the duration of the program. The remaining three classes of storage define dynamic storage allocation processes that depend on factors within the program.

All variables not declared to have a storage class are defined, by default, as automatic. A variable of the AUTOMATIC storage class is allocated storage upon activation of the block in which the variable is declared, but loses this allocation – and its value – when the block is terminated. Thus, storage is allocated only as it is needed, and is freed when no longer needed, reducing the amount of storage required and relieving the programmer of tasks associated with storage management.

The programmer is allowed total control of storage allocation when a variable is declared CONTROLLED. The programmer allocates storage and frees storage for each variable, as desired. Further, with a controlled variable, the programmer can allocate storage for a variable even when a previous allocation for that variable exists, creating a pushdown stack for that variable. Previous allocations are not released; when the top allocation is freed, the next allocation becomes available. These stacked allocations remain even after the block in which they are declared is terminated. Any reference to a stacked controlled variable receives the latest allocation.

The highest degree of programmer control of storage allocation for variables is the BASED class; however, it is also the most difficult to use properly. Based storage can be allocated and freed, and more than one allocation can exist for a variable, just as with controlled storage. The major difference is that each of the current based allocations is available at any time by use of a pointer value.

### RECURSION

An active procedure can be reactivated from within itself or by a call from another active procedure. When this occurs, values of variables declared automatic are saved in an environment similar to the pushdown stack of controlled storage.

## CONDITIONS

During execution of a PL/I program, exceptional conditions (such as errors, special status, end-of-file, and so forth) are detected when they occur. Each of these conditions has a keyword name to aid the programmer in handling exceptional conditions. Most conditions cause an interrupt when they occur, which allows the system to handle them in a predefined manner. Many of the conditions can be enabled or disabled by the programmer to allow or override such an interrupt.

The PL/I programmer must be familiar with the types of data permitted, the organization of the data, and the methods available for referring to data. All data in a PL/I program can be considered either computational data used to represent values to be processed, or noncomputational data used by the programmer to control execution of the program.

The variety of data types that can be specified and used contribute greatly to the range of applications for which PL/I can be used.

## DATA TYPES

Computational data consists of three general types:

Arithmetic data

String data

Pictured data

Arithmetic data items consist of numeric values that have the characteristics of base, scale and precision. These characteristics specify whether the value is decimal or binary, fixed or floating point, the size of the value in digits, and the placement of the decimal point.

String data consists of sequences of characters or bits.

Pictured data is similar to pictured data in COBOL.

Noncomputational data can be of six types:

Label

Area

File

Entry

Pointer

Offset

In addition to the variety of data types available, automatic conversion from one data type to another, in most cases on an as needed basis, is an intrinsic part of the language. This conversion feature, in almost all cases, ensures that a program will run to a satisfactory conclusion rather than abort because of incompatible data types.

## DATA AGGREGATES

PL/I supports arrays and structures. The arrays are similar to FORTRAN arrays. The structures are analogous to COBOL group items.

### ARRAYS

Data elements with the same data type and of the same precision or length can be grouped together to form an array. When the array is created, it is given a name and is considered a unit. A subsequent reference to an item in the array can be made only by array name and the relative position of the element in the array. Arrays can have as many as 32 dimensions.

### STRUCTURES

Data items that possess a logical relationship to one another, but do not necessarily have identical characteristics, can be grouped into structures. The entire structure is given a name by which it can be referenced; however, unlike an array, each element of a structure must also have a name.

Structures are subdivided into units of different levels, allowing hierarchies to be created. A structure can have up to 63 levels.

It is possible to declare an array as an element of a structure. It is also possible to create an array of structures.

## DATA STORAGE

Allocation of storage was often a problem to programmers writing in the high-level languages which preceded PL/I. One of the problems encountered was the assignment of storage each time a variable was declared. This often resulted in the assignment of more storage than required, or worse, more than was available. In PL/I, the programmer is greatly relieved of the task of allocating storage, and storage management is simplified through the application of concepts not previously available.

As explained in section 3, PL/I allows four ways of assigning data storage. STATIC storage is assigned when a job is loaded and remains assigned for the length of the job. AUTOMATIC storage is assigned upon entry to the block in which it is declared, and released upon exit from that block. CONTROLLED storage is explicitly assigned and released by the programmer. BASED storage is similar to CONTROLLED storage, but allows concurrent references to each assigned storage area.

In addition to storage control, PL/I allows special handling of structures and arrays through the use of special keyword attributes. The DEFINED attribute specifies that a named element, structure, or array is to occupy the same storage location as that assigned to other data. It can also be used to subdivide or overlay one data item on another. Other keyword attributes allow the programmer to create a new structure based on the attributes of a higher level structure, to specify the positioning in storage of data elements, and to specify initial values of variables.

## BUILT-IN FUNCTIONS

PL/I includes a comprehensive set of predefined functions as part of the language. These built-in functions include not only the commonly used arithmetic functions but also other useful functions related to language facilities, such as functions for manipulating strings and arrays.

Built-in functions are invoked in the same manner as functions defined by the programmer. A built-in function's action is automatically performed when the built-in function name is encountered in the execution of the program.

The built-in functions are included in the list of keywords in appendix A.

CYBER PL/I input/output processing is handled by CYBER Record Manager. CYBER Record Manager is a standard Control Data software product that supplies a group of interface routines between user programs, and operating system routines that read and write files on the input/output devices. CYBER Record Manager provides common file and record formats, thus promoting interchangeability of information between higher level programming languages (such as PL/I, FORTRAN, and COBOL) and between the NOS and NOS/BE operating systems. The CYBER PL/I run-time modules provide the interface required to translate a file input/output statement into an appropriate CYBER Record Manager dataset input/output request.

## FILES AND DATASETS

A dataset is a collection of information that is not part of a PL/I program, but can be accessed by the program. The dataset can reside on any storage medium available to the PL/I program.

A file, in PL/I, is the entity through which data can be channeled during program execution. In PL/I, a file can be associated with different datasets at different times during program execution. The files defined in a compiled PL/I program have no direct correlation to specific datasets. The linkage between a file and a dataset is made only when the file is opened during program execution.

A file name is the PL/I identifier with which the programmer declares and refers to a file.

A dataset name is used by the executing program to access data resident in auxiliary storage. The dataset name, not the file name, is the name known to CYBER Record Manager and to the operating system. The dataset name in PL/I corresponds to the logical file name used by CYBER Record Manager and the rest of the CYBER operating system and its related software products.

When a file is opened, the interface between it and a particular dataset is established. This association remains in effect until the file is closed.

## INPUT/OUTPUT TRANSMISSION

Two types of input/output transmission are available to the CYBER PL/I user, record transmission and stream transmission.

Record transmission is the input or output of whole records, during which no conversions occur. Declaration of the record attribute for a file ensures that the records remain intact; such files can be accessed by the following statements:

READ

WRITE

REWRITE

DELETE

LOCATE

Stream transmission is the input or output of data as a continuous flow of characters. The characters in the stream are converted automatically on input to conform to the variables to which they are assigned; on output, they are automatically converted to character representation. The stream is usually organized in lines. The programmer can direct the editing and format of the data as well as the organization of data items into lines.

Declaration of the stream attribute for a file enables the programmer to use GET and PUT statements to transmit data from and to datasets or internal storage. Stream input/output statements options include:

FILE

STRING

LINE

COPY

PAGE

SKIP

The major difference between record transmission and stream transmission is that stream input/output creates datasets intended to be legible to the user, whereas record input/output is intended for efficient input/output transmission and, therefore, utilizes internal machine forms.

Stream data formatting is simplified for the user familiar with FORTRAN or COBOL because both a FORMAT statement and a PICTURE attribute are available to the programmer. The PL/I programmer can choose either method, depending on the characteristics of the data to be manipulated.

The following is an alphabetical list of keywords, including built-in function names, and acceptable abbreviations. Keywords and their abbreviations are not reserved words; they have special meaning only in the contexts noted.

| Keyword | Context |
|---|---|
| A | Format item |
| ABS | Built-in function |
| ACOS | Built-in function |
| ADD | Built-in function |
| ADDR | Built-in function |
| AFTER | Built-in function |
| ALIGNED | Attribute |
| ALLOCATE/ALLOC | Statement identifier |
| ALLOCATION /ALLOCN | Built-in function |
| AREA | Attribute; condition |
| ASIN | Built-in function |
| ATAN | Built-in function |
| ATAND | Built-in function |
| ATANH | Built-in function |
| AUTOMATIC/AUTO | Attribute |
| B | Format item |
| BASED | Attribute |
| BEFORE | Built-in function |
| BEGIN | Statement identifier |
| BINARY/BIN | Attribute; built-in function |
| BIT | Attribute; built-in function |
| BOOL | Built-in function |
| BUILTIN | Attribute |
| BY | Statement option (DO; do-specification in GET, PUT) |
| CALL | Statement identifier |
| CEIL | Built-in function |
| CHARACTER/CHAR | Attribute; built-in function |
| CLOSE | Statement identifier |
| COLLATE | Built-in function |
| COLUMN/COL | Format item |
| CONDITION/COND | Condition |
| CONTROLLED/CTL | Attribute |
| CONVERSION/CONV | Condition; enabled-condition-name |
| COPY | Statement option (GET); built-in function |
| COS | Built-in function |
| COSD | Built-in function |
| COSH | Built-in function |
| DATE | Built-in function |
| DECAT | Built-in function |
| DECIMAL/DEC | Attribute; built-in function |
| DECLARE/DCL | Statement identifier |
| DEFINED/DEF | Attribute |
| DELETE | Statement identifier |
| DIMENSION/DIM | Built-in function |
| DIRECT | Attribute; statement-option (OPEN) |
| DIVIDE | Built-in function |
| DO | Statement identifier; statement-option (GET, PUT) |
| DOT | Built-in function |
| E | Format item |
| EDIT | Statement option (GET, PUT) |
| ELSE | Statement option (IF) |
| EMPTY | Built-in function |
| END | Statement identifier |
| ENDFILE | Condition |
| ENDPAGE | Condition |
| ENTRY | Attribute; statement identifier |
| ENVIRONMENT/ENV | Attribute; statement-option (CLOSE, OPEN) |
| ERF | Built-in function |
| ERFC | Built-in function |
| ERROR | Condition |
| EXP | Built-in function |
| EXTERNAL/EXT | Attribute |
| F | Format item |
| FILE | Attribute; statement option (input/output statements) |
| FINISH | Condition |
| FIXED | Attribute; built-in function |
| FIXEDOVERFLOW /FOFL | Condition; enabled-condition-name |
| FLOAT | Attribute; built-in function |
| FLOOR | Built-in function |
| FORMAT | Statement identifier |
| FREE | Statement identifier |
| FROM | Statement option (WRITE, REWRITE) |
| GET | Statement identifier |
| GOTO/GO TO | Statement identifier |
| HBOUND | Built-in function |
| HIGH | Built-in function |
| IF | Statement identifier |
| IGNORE | Statement option (READ) |
| IN | Statement option (ALLOCATE, FREE) |
| INDEX | Built-in function |
| INITIAL/INIT | Attribute |
| INPUT | Attribute; statement option (OPEN) |
| INTERNAL/INT | Attribute |
| INTO | Statement option (READ) |
| KEY | Condition; statement option (READ, DELETE, REWRITE) |
| KEYED | Attribute; statement option (OPEN) |
| KEYFROM | Statement option (WRITE, LOCATE) |
| KEYTO | Statement option (READ) |
| LABEL | Attribute |
| LBOUND | Built-in function |
| LENGTH | Built-in function |
| LIKE | Attribute |
| LINE | Statement option (PUT); format item |
| LINENO | Built-in function |
| LINESIZE | Statement option (OPEN) |
| LIST | Statement option (GET, PUT) |

| Keyword | Context |
|---|---|
| LOCATE | Statement identifier |
| LOG | Built-in function |
| LOG10 | Built-in function |
| LOG2 | Built-in function |
| LOW | Built-in function |
| MAIN | Option in OPTIONS-option |
| MAX | Built-in function |
| MIN | Built-in function |
| MOD | Built-in function |
| MULTIPLY | Built-in function |
| NOCONVERSION /NOCONV | Disabled-condition-name |
| NOFIXEDOVERFLOW/ NOFOFL | Disabled-condition-name |
| NOOVERFLOW/NOOFL | Disabled-condition-name |
| NOSIZE | Disabled-condition-name |
| NOSTRINGRANGE /NOSTRG | Disabled-condition-name |
| NOSUBSCRIPTRANGE/ NOSUBRG | Disabled-condition-name |
| NOUNDERFLOW/ NOUFL | Disabled-condition-name |
| NOZERODIVIDE /NOZDIV | Disabled-condition-name |
| NULL | Built-in function |
| OFFSET | Attribute; built-in function |
| ON | Statement identifier |
| ONCHAR | Built-in function; pseudovariable |
| ONCODE | Built-in function |
| ONFILE | Built-in function |
| ONKEY | Built-in function |
| ONLOC | Built-in function |
| ONSOURCE | Built-in function; pseudovariable |
| OPEN | Statement identifier |
| OPTIONS | Statement-option (PROCEDURE) |
| OUTPUT | Attribute; statement-option (OPEN) |
| OVERFLOW/OFL | Condition; enabled-condition-name |
| P | Format item |
| PAGE | Statement option (PUT); format item |
| PAGENO | Built-in function; pseudovariable |
| PAGESIZE | Statement option (OPEN) |
| PICTURE/PIC | Attribute |
| POINTER/PTR | Attribute; built-in function |
| POSITION/POS | Attribute |
| PRINT | Attribute; statement option (OPEN) |
| PROCEDURE/PROC | Statement identifier |
| PROD | Built-in function |
| PUT | Statement identifier |
| R | Format item |
| READ | Statement identifier |
| REAL | Attribute |
| RECORD | Attribute; condition; statement-option (OPEN) |
| RECURSIVE | Statement option (PROCEDURE) |
| REFER | Extent-expression option |
| RETURN | Statement identifier |
| RETURNS | Attribute; statement option (PROCEDURE, ENTRY) |

| Keyword | Context |
|---|---|
| REVERSE | Built-in function |
| REVERT | Statement identifier |
| REWRITE | Statement identifier |
| ROUND | Built-in function |
| SEQUENTIAL/SEQL | Attribute; statement-option (OPEN) |
| SET | Statement option (ALLOCATE, LOCATE, READ) |
| SIGN | Built-in function |
| SIGNAL | Statement identifier |
| SIN | Built-in function |
| SIND | Built-in function |
| SINH | Built-in function |
| SIZE | Condition; enabled-condition-name |
| SKIP | Statement option (GET,PUT); format item |
| SNAP | Statement option (ON) |
| SQRT | Built-in function |
| STATIC | Attribute |
| STOP | Statement identifier |
| STORAGE | Condition |
| STREAM | Attribute; statement-option (OPEN) |
| STRING | Statement option (GET, PUT) |
| STRINGRANGE/STRG | Condition; enabled-condition-name |
| SUBSCRIPTRANGE /SUBRG | Condition; enabled-condition-name |
| SUBSTR | Built-in function; pseudovariable |
| SUBTRACT | Built-in function |
| SUM | Built-in function |
| SYSTEM | Statement option (ON) |
| TAN | Built-in function |
| TAND | Built-in function |
| TANH | Built-in function |
| THEN | Statement option (IF) |
| TIME | Built-in function |
| TITLE | Statement option (OPEN) |
| TO | Statement option (DO; do-specification in GET, PUT) |
| TRANSLATE | Built-in function |
| TRANSMIT | Condition |
| TRUNC | Built-in function |
| UNALIGNED/UNAL | Attribute |
| UNDEFINEDFILE /UNDF | Condition |
| UNDERFLOW/UFL | Condition; enabled-condition-name |
| UNSPEC | Built-in function; pseudovariable |
| UPDATE | Attribute; statement-option (OPEN) |
| VALID | Built-in function |
| VARYING/VAR | Attribute |
| VERIFY | Built-in function |
| WHILE | Statement option (DO; do-specification in GET, PUT) |
| WRITE | Statement identifier |
| X | Format item |
| ZERODIVIDE/ZDIV | Condition; enabled-condition-name |

## COMMENT SHEET

CONTROL DATA CORPORATION

TITLE: CYBER PL/I
General Information Manual

PUBLICATION NO. 60388300 REVISION A

This form is not intended to be used as an order blank. Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

FROM NAME: _____ POSITION: _____

COMPANY NAME: _____

ADDRESS: _____

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

**BUSINESS REPLY MAIL**
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

**CONTROL DATA CORPORATION**
*Publications and Graphics Division*
**215 Moffett Park Drive**
**Sunnyvale, California 94086**

CUT ON THIS LINE

FOLD

FOLD

STAPLE

STAPLE