SC33-0029-3
File No. S360/S370-29

Program Product

OS
PL/I Optimizing Compiler:
TSO User's Guide

IBM

SC33-0029-3
File No. S360/S370-29

Program Product

OS
PL/I Optimizing Compiler:
TSO User's Guide

Program Numbers **5734-PL1**
5734-LM4
5734-LM5

(These program products are available as composite package 5734-PL3)

IBM

FOURTH EDITION (July 1979)

This is a major revision of, and makes obsolete, SC33-0029-2, and its technical newsletter, SN33-6163.

This edition applies to Version 1, Release 3, Modification 1, of the OS PL/I Optimizing Compiler and libraries, Program Products 5734-PL1, 5734-LM4, and 5734-LM5, and to any subsequent version, release, and modification until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Amendments" following the list of figures. Technical changes made are indicated by a vertical bar to the left of the change. These bars will be deleted in any subsequent republication of the affected page. Editorial changes that have no technical significance are not noted.

Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 Bibliography, GC20-0001, for the editions that are applicable and current.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming or services which are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming or services in your country.

Publications are not stocked at the address given below; requests for 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 Corporation, P. O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

This book is an extension of the OS PL/I Optimizing Compiler: Programmer's Guide. It describes how to use the facilities of TSO to compile a PL/I program using the PL/I Optimizing Compiler, and how to execute the program. The reader is assumed to be familiar with PL/I and TSO; the manual covers both the OS/MVT and the OS/VS2 versions of TSO.

Requisite publications for this manual are listed below:

OS PL/I Optimizing and Checkout Compilers: Language Reference Manual, GC33-0009

OS PL/I Optimizing Compiler: Programmer's Guide, SC33-0006

IBM System/360 Operating System: Time Sharing Option Command Language Reference, GC28-6732

OS/VS2 TSO Command Language Reference, GC28-0646

IBM System/360 Operating System: Time Sharing Option Terminal User's Guide, GC28-6763

OS/VS2 TSO Terminal User's Guide, GC28-0645

OS/MVT and OS/VS2 TSO Terminal Manual, GC28-6762

The terminal user's guides are the basic authoritative sources for information about the PL/I command and about the compiler options under TSO. The authoritative sources of information on other topics are as follows:

- TSO facilities other than PLI command: the Command Language Reference
- Terminals: the Terminal Manual
- The PL/I Optimizing Compiler: The Programmer's Guide
- The PL/I language implemented by the PL/I Optimizing Compiler: the PL/I Language Reference Manual

The authoritative sources will be the first to be updated to reflect any changes.

Contents

Figures

Summary of Amendments

JULY 1979

SERVICE CHANGES

For Release 3, Modification 1, the storage size in which the compiler runs has been increased.

OTHER MODIFICATIONS

This publication has been redesigned into one part with six chapters and three appendixes.

Chapter 1 remains an introduction giving an overview of the advantages of using TSO to create, compile, and execute data sets.

Information that was formerly contained in Chapter 2, "Operating the Terminal," and Chapter 3 "The Terminal Session," can be found in the TSO Terminal User's Guide for your installation.

Chapter 2 (formerly Chapter 4) has been revised and condensed and gives a brief discussion on creating and updating PL/I programs. The TSO Command Language Reference manual for your installation contains detailed information on this subject.

Chapter 3 (formerly Chapter 5) is a description of how to compile and execute a PL/I program under TSO.

Chapter 4 (formerly Chapter 6) discusses input and output procedures and attention interrupts. Detailed information on allocating and performing operations on data sets can be found in the TSO Command Language Reference manual for your installation.

Chapter 5 (formerly Part 2) describes the PL/I command that is used to invoke the Optimizing Compiler under TSO.

Chapter 6 (formerly Part 2) describes compiler options.

Appendix A briefly discusses command syntax under TSO.

Appendix B has been condensed and describes items relevant to PL/I when you use a terminal other than an IBM 2741 Communications Terminal. Information on the IBM 2260 and 2265 Display Stations can be found in the appropriate manuals for those terminals.

Former Appendixes C through E have been deleted. Information on data set types (formerly included in Appendix D) has been moved to Chapter 2; naming conventions for data sets (formerly in Appendix E) are discussed in the TSO Terminal User's Guide for your installation.

Appendix C (formerly Appendix F) discusses creating secondary data sets with the %INCLUDE statement.

Former Appendix G has been deleted. Information about programs of non-TSO data sets can be found in the TSO Command Language Reference manual for your installation.

Chapter 1: Introduction

TSO

The facilities of TSO that will be of particular interest to a PL/I programmer are those for:

- Creating, updating, and manipulating data sets
- Invoking programs
- These facilities are described in your TSO User's Guide.

The data sets can hold PL/I programs and input data for them. The program-invoking facilities can be used to invoke PL/I compilers (including the optimizing compiler), load modules, and OS facilities such as the linkage editor and the loader.

CONVERSATIONAL PROCESSING

The optimizing compiler operates under TSO in conversational mode.

In conversational mode, you invoke each program you wish to run at the time you wish to execute it. Execution generally starts almost as soon as you send the instruction from the terminal, and results can be displayed at your terminal as soon as the program produces them. If the program requires input data, you can enter it at the terminal whenever the execution reaches an appropriate input statement.

A benefit of TSO is the ability it gives you to interact conversationally not just with your own programs, but with the computing system as a whole. Nearly all the operations a programmer needs to perform can be carried out from a terminal.

THE OPTIMIZING COMPILER

The optimizing compiler will compile PL/I source programs, print the diagnostic messages at the terminal, and write the object modules onto a data set. These object modules can then be conversationally link-edited and executed.

During execution, the terminal can be used as an input and output device for consecutive files in the program. You can therefore receive output from your program and provide input for it. Conversational I/O needs no special PL/I code, so any STREAM file can be used conversationally.

The INTERRUPT compiler option, introduced with Release 3 of the compiler, allows attention interrupts to become an integral part of programs compiled on the optimizing compiler. Chapter 6 has more information on this option.

When you create a PL/I program using the EDIT command, you must specify the data set type as either PLI or PLIF. PLI indicates that the records are to be of variable length; PLIF indicates that they are to be of fixed length. Both types are accepted by the PL/I optimizing and checkout compilers.

You should distinguish carefully between the data set type and the descriptive qualifier in the data set name. Both PLI and PLIF data set types take the descriptive qualifier PLI.

The following figure lists some descriptive qualifiers and the data types they describe.

Data	Descriptive Qualifier
PL/I source code	PLI
Object module	OBJ
Formatted source code	FORM
Compiler listing of source code	LIST
Preprocessor output in card format	DECK

Figure 1. Some descriptive qualifiers

The differences between the two types of data set are as follows:

1. Record formats. You can control the format by means of the LINE and BLOCK operands of the EDIT command (see the manual TSO Command Language Reference), but the standard default formats are as follows.

   PLI-type:
   Record format: VB
   Maximum logical record length: 104 bytes
   Block size: 500 bytes

   PLIF-type:
   Record format: FB
   Logical record length: 80 bytes
   Block size: 400 bytes

2. Location of sequence numbers. In PLI-type records, sequence numbers are held in bytes 1 to 8; in PLIF-type, they are in bytes 73 to 80.

You can have line numbers omitted from the records by specifying the additional operand NONUM. In this book, however, it is assumed that the records have line numbers.

Provided you allow the standard defaults for the MARGINS, SEQUENCE, NUMBER, and STMT compiler options to apply when you compile your program, the compiler will assume all source text (including *PROCESS statements) starts after column 8 for V-format data sets and extends from columns 2

to 72 for F-format data sets. The statement numbers printed on all compiler listings will be derived from the sequence numbers.

The default options for each field are summarized in Figure 2.

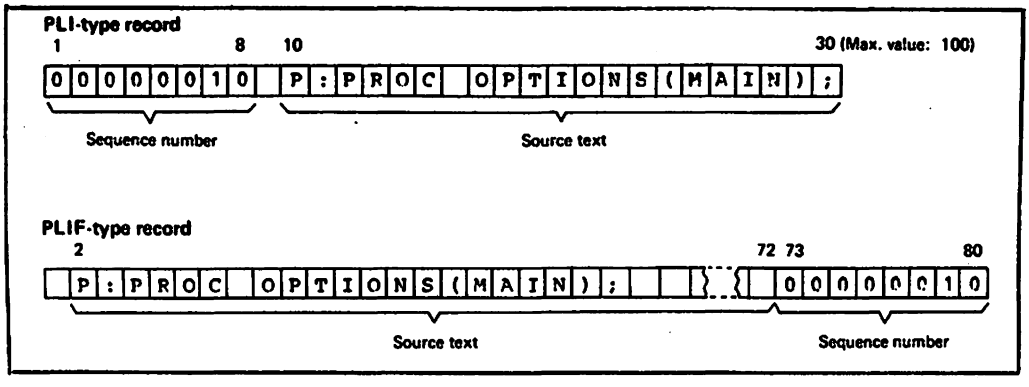


Figure 2. Standard default record formats

The differences between the record types are shown in Figure 2. As that figure suggests, PLI-type data sets generally take less space to hold a particular size of program than PLIF-type.

Notice that although the standard defaults assume no carriage control character for PLI-type records, byte 9 is assumed to be occupied by neither source text nor sequence number. It can therefore be used to hold a carriage control character if there is any possibility of the program being listed in batch mode.

COMPILATION

INVOKING THE COMPILER

The usual method of invoking the optimizing compiler is by the PLI command. In its simplest form, the command consists of the keyword and the name of the TSO data set holding the PL/I source program. For instance:

```
pli caltrop
```

In addition to the data set name, you can also specify the PRINT operand to control the compiler listings, and the LIB operand to specify secondary input data sets for %INCLUDE statements. You can also specify compiler options as operands of the PLI command.

The command processor for the PLI command is a program known as the PL/I prompter. When the command is entered, this program checks the operands and allocates the data sets required by the compiler. Control is then passed to the compiler and a message displayed.

If the source data set has a conventional TSO data set name, you can use the simple name, as in the example. If not, you will need to specify the full name, and enclose it in single quotation marks:

```
pli 'dianthus'
```

or:

```
pli 'jjones.erica.pli'
```

The compiler translates the source programs into object modules, which it stores on external data sets. You can link-edit and execute these object modules conversationally (see "Link Editing and Executing" later in this chapter).

If you use an unqualified data set name, as in the example at the start of this section, the system will generate a name for the object module data set. It will take the simple name of the source data set - CALTROP in the example - and add to it your user-identification and the descriptive qualifier OBJ. Hence, if the user who entered the example PLI command had the identification WSMITH, the object module would be written onto a data set called WSMITH.CALTROP.OBJ.

You can make your own choice of name for the object module data set by including the OBJECT compiler option as an operand of the PLI command. For instance:

```
pli caltrop object(trapa)
```

The system will add the same qualifiers to this name as it would to the source data set simple name, so the object module would, in this example, be written onto a data set called WSMITH.TRAPA.OBJ.

You can specify the full name of the object module data set by enclosing it in quotation marks. For instance:

```
pli caltrop object('natans')
```

The system in this case adds no qualifiers, so the object module is stored on a data set called NATANS.

The specification of a full name can be used to store the object module with another user's user-identification. For instance, the following command would store the object module using the user-identification JJONES:

```
pli caltrop object('jjones.caltrop.obj')
```

An alternative to the PLI command is the RUN command or subcommand.

RUN COMMAND AND SUBCOMMAND

The RUN command is used to invoke a specified compiler and to execute the compiled program. It can be used to invoke, via the prompter, the optimizing compiler. Compared with the PLI command, the RUN command has the following limitations:

- No compiler options can be specified.
- There is no PRINT operand, so if your system was generated with the IBM default for this operand, the prompter will allocate SYSPRINT to a dummy data set. You cannot therefore obtain compiler listings at the terminal.
- Under the OS/MVT version of TSO, there is no LIB operand, so you cannot specify a preprocessor input data set. Such a data set must be allocated by means of the ALLOCATE command. (The enhanced OS/VS2 version includes the LIB operand.)
- The source program data set must have V-format records. A data set created under the EDIT command is therefore suitable if the PLI operand was specified, but not if the PLIF operand was used.

A simplified general form of the command is:

```
RUN|R data-set-name ['parameter-string']
      [PLI] [CHECK|OPT]
```

The "data-set-name" operand must be the name of the data set holding the PL/I source program. It can be specified in the form "user-supplied-name.descriptive-qualifier". If it is not specified in this form - if, for instance, only the simple name is specified - or if the descriptive qualifier is not PLI, you must include the operand PLI. The CHECK and OPT operands indicate, respectively, the checkout and optimizing compiler; OPT is the standard default applied when neither is specified. The "parameter-string" will be passed to the main procedure of the PL/I program.

RUN can also be used as a subcommand of the EDIT command. The syntax shown here for the RUN command also applies to the subcommand, except that there is no "data-set-name" or PLI operand. Approximately 30K bytes of storage will be reserved in your region for the EDIT command routines, so the storage available to the compiler and your program will be reduced by this amount. RUN is particularly useful as a subcommand

when a program is being developed. After a run, amendments can be made to the source data set by means of other subcommands of EDIT, without the need to reenter the EDIT command.

COMPILER LISTINGS

As in batch mode, compiler options control which listings the compiler produces (see "Compiler Options" in Chapter 6). You can specify the options as operands of the PLI command.

In addition to specifying which options are to be produced, you will need to indicate where they are to be transmitted to. If you wish to have them displayed at the terminal, you can specify either the PRINT(*) operand, which allocates the compiler listing file to the terminal, or the TERMINAL option. The latter should contain a list of the options corresponding to the listings you require at the terminal. For instance, to produce a source listing at the terminal, you could enter either:

```
pli caltrop print(*) source
```

or:

```
pli caltrop term(source)
```

Compiler listings can be directed to a data set by specifying the PRINT operand with the data set's name, or to a SYSOUT class by specifying the SYSPRINT operand. For further details see "PLI Command" in Chapter 5.

CALLING THE COMPILER

The CALL command can be used to invoke the compiler directly. The member name is IEL0AA and you call the entry point IEL0AC. Because the PL/I prompter is not used, you must ensure that all data sets required by the compiler are allocated before it is called (see "Compiler Data Sets" in Chapter 4). The command takes the form:

```
CALL IEL0AC
```

BACKGROUND PROCESSING

If you have the necessary authorization, you can submit jobs for processing in a background region. Your installation must record the authorization in your UADS (User Attribute Data Set) entry.

Jobs are submitted by means of the SUBMIT command. The command must include the name of the data set holding the job or jobs to be processed, and the data set must contain the necessary Job Control Language statements. Jobs will be run under the same version of the operating system as is used for TSO. Output from the jobs can be manipulated from your terminal.

Further details about submitting background jobs are given in the manual TSO Terminal User's Guide.

LINK-EDITING AND EXECUTING

INTRODUCTION TO COMMANDS

Compilation using the PLI command produces an object module on a data set; to execute the program this must be link-edited to form a load module, which must then be invoked. These operations can be performed by means of either the LOADGO command, or the LINK command followed by the CALL command.

The LOADGO command invokes the loader program. This builds the load module in main storage, and then invokes it. The load module can be built from a single load module, or from a number of object modules or load modules, or a mixture of object and load modules.

The LINK command invokes the linkage editor, which link-edits one or more object modules into a load module. It stores the load module on an external data set. The CALL command invokes the load module.

You can use the LOADGO method if you do not wish to retain a copy of the load module. Otherwise, you must use the LINK-and-CALL method.

An alternative to the LOADGO method is the RUN method, in which the single command or subcommand RUN initiates compilation followed by link-editing and execution. The RUN command and subcommand are described in the earlier section "RUN Command and Subcommand".

The succeeding sections describe the LOADGO, LINK, and CALL commands.

THE LOADGO COMMAND

This command invokes the OS loader. A simplified general form is:

```
LOADGO|LOAD (data-set-list) ['parameter-string']
            [LIB(data-set-list)] [PLICMIX|PLIBASE]
```

The initial "data-set-list" specifies the names of data sets holding the modules - object modules or load modules - that are to be loaded and executed. The names must be separated by commas or blanks; the parentheses can be omitted when there is only one name.

For programs compiled on the optimizing compiler "parameter-string" contains two fields separated by a slash (/), and takes the form:

```
'[execution time options][/parameter string]'
```

The first field will be passed to the library initialization routine as an execution-time option list; the second will be passed to the main procedure of the PL/I program. If there is no list of execution time options, the main procedure parameter must be preceded by a /. Execution time options are described in the manual OS PL/I Optimizing Compiler: Programmer's Guide.

The LIB operand should contain a list of the names of data sets that contain user-supplied modules to be link-edited by means of the automatic library call facility.

The name of the PL/I resident library must be specified as an operand. It can be either PLIBASE or PLICMIX. PLIBASE must be used in normal conditions. PLICMIX is for use when modules that have been translated by the checkout compiler are being mixed with those from the optimizing compiler. In such conditions, PLICMIX gives a smaller load

module than the use of PLIBASE, although execution may be slower.

You can specify loader options as operands of the LOADGO command; for further information, see the OS/VS2 TSO: Command Language Reference manual.

The LOADGO command processor allocates all data sets used by the loader.

THE LINK COMMAND

This command invokes the OS linkage editor. A simplified general form is:

```
LINK (data-set-list) [LOAD(data-set-name)]
     [LIB(data-set-list)] [PLICMIX|PLIBASE]
```

The initial "data-set-list" specifies the names of data sets holding the object modules that are to be link-edited. They can also contain linkage-editor control statements. The names must be separated by commas or blanks; the parentheses can be omitted when there is only one name. The rules about positioning of control statements in relation to object modules are the same as for batch mode.

The LOAD operand specifies the name of the data set that is to hold the load module. If you specify a simple name, the system will add your user-identification qualifier and the descriptive qualifier LOAD. The load module must be a member of a partitioned data set. If you do not specify a member name, the system will use the name TEMPNAME. If you omit the LOAD operand, the system will construct a name by adding your user-identification qualifier and the descriptive qualifier LOAD to the first data set name in the initial "data-set-list".

The LIB operand should contain a list of the data sets that contain user-supplied modules to be link-edited by means of the automatic library call facility.

The name of the PL/I resident library must be specified. It can be either PLIBASE or PLICMIX. PLIBASE must be used in normal conditions. PLICMIX is for use when modules that have been translated by the checkout compiler are being mixed with those from the optimizing compiler. In these conditions, either PLICMIX or PLIBASE can be used. PLICMIX produces a smaller load module, although execution may be slower than if PLIBASE is used.

You can also specify linkage editor options in the LINK command; for further information, see the OS/VS2 TSO: Command Language Reference manual.

The LINK command processor allocates all data sets required by the linkage editor.

THE CALL COMMAND

This command loads and executes a specified load module. Its general form is:

```
CALL data-set-name ['parameter-string']
```

The "data-set-name" specifies the partitioned data set member that holds the load module. If you specify the simple name of the data set, the system assumes the descriptive qualifier LOAD. If you do not specify a member name, the system assumes the name TEMPNAME.

For programs compiled on the optimizing compiler, "parameter-string"

contains two fields separated by a slash(/), and takes the form:

'[execution time options][/parameter string]

The first field will be passed to the library initialization routine as an execution-time option list; the second will be passed to the main procedure of the PL/I program. If there is no list of execution time options, the main procedure parameter must be preceded by a /. Execution time options are described in the manual OS PL/I Optimizing Compiler: Programmer's Guide.

MIXING PROCEDURES FROM THE PL/I OPTIMIZING AND CHECKOUT COMPILERS

Procedures processed by the two compilers can be link-edited to form a single program. The LOADGO or LINK command is used to do this, in the same way as for procedures from the optimizing compiler alone. However, the following special points should be noted:

- The data set holding the checkout compiler and a work file must be allocated. This can be done by the LOGON procedure. The IBM standard logon procedure for the checkout compiler, PLICKLGN, makes the allocations, but that for the optimizing compiler, PLIXLGN, does not. PLICKLGN is suitable for use with the optimizing compiler. As an alternative to the logon procedure, ALLOCATE commands can be used. For instance:

  ```
  ALLOCATE FILE(SYSPLIC) DATASET('SYS1.PLICLNK') SHR
  ALLOCATE FILE(SYSUT1) BLOCK(1024) SPACE(60,60)
  ```

  The SHR operand is required to allow other users to access the compiler.

- The checkout compiler must have control at the start of execution. Control must not pass initially to a procedure compiled by the optimizing compiler. One way of ensuring this is to put a data set holding a procedure translated by the checkout compiler at the start of the list of data sets in the LOADGO or LINK command.

- The PL/I resident library PLICMIX can be used as an alternative to PLIBASE. The required version can be specified in the LOADGO or LINK command. PLICMIX gives a smaller load module at the expense of execution time; PLIBASE gives faster execution at the expense of space.

Further information about mixing modules from the two compilers is given in the programmer's guide for the checkout compiler. Examples are given in the following two sections.

COMPILE, LOAD, AND GO

You should use the load-and-go method when you wish to:

- Build the load module in main storage, without producing a permanent copy on a data set, and either:

- Link-edit two or more external procedures together and execute the resulting program, or:

- Execute a single external procedure

If you want a permanent copy of the load module, you must use the LINK command, examples of which are given later.

Further information about using the loader is given in the programmer's guide for the optimizing compiler.

Example 1

A single external procedure, held on a data set called PGROUP3.ZA311B.PLI, is compiled by the optimizing compiler, and the resulting code is link-edited and executed by means of the LOADGO command.

The terminal session is started by a LOGON command specifying a user-identification of PGROUP3. The compiler is then invoked by means of a PLI command. No data set name is specified for the object module data set, so the system generates one from the name of the source data set: the object module data set will be called PGROUP3.ZA311B.OBJ.

```
logon pgroup3
PGROUP3 LOGON IN PROGRESS AT 13:41::07 ON FEBRUARY 7, 1979
READY
pli za311b
PL/I OPTIMIZER V1 R2.0 TIME: 13.41.08 DATE: 7 FEB 79


NO MESSAGES PRODUCED FOR THIS COMPILATION
COMPILE TIME     0.02 MINS       SPILL FILL:     0 RECORDS, SIZE  4051
READY
loadgo za311b plibase
READY
```

Example 2

An external procedure that uses the procedure compiled in Example 1 as a subroutine is compiled and executed. The two procedures are link-edited together and executed by means of the LOADGO command. The main procedure is held on a data set called PGROUP3.XA311B.PLI, and it is first compiled by the optimizing compiler. It is then link-edited with the object module on PGROUP3.ZA311B.OBJ, produced in Example 1. The OBJECT option specifies XAB as the simple name for the object module data set. The full name of the data set will be PGROUP3.XAB.OBJ.

The parameter '999' is passed to the main procedure at execution time.

```
pli xa311b obj(xab)
PL/I OPTIMIZER V1 R2.0 TIME: 13.52.41  DATE: 7 FEB 79
OPTIONS SPECIFIED
 OBJ


NO MESSAGES PRODUCED FOR THIS COMPILATION
COMPILE TIME     0.02 MINS       SPILL FILE:       0 RECORDS, SIZE  4051
READY
loadgo (xab,za311b) '/999' plibase
READY
```

If the execution time option REPORT were to be specified, the LOADGO command would take the form:

```
loadgo (xab,za311b) 'report/999' plibase
```

Example 3

A procedure compiled by the optimizing compiler uses, as a subroutine, a procedure that was translated by the PL/I checkout compiler. The main procedure is held in object-module form on a data set whose full name is REVERT; this name is not in accordance with the TSO conventions. The translated version of the subroutine is held on two data sets: PGROUP3.TOMPROC.OBJ, which holds the link-edit stub, and PGROUP3.TOMPROC.ITEXT, which holds the intermediate text. The latter data set must be allocated before the LOADGO command is entered. The data set holding the checkout compiler, SYS1.PLICLINK, and a work file, called SYSUT1, must also be allocated unless this is done in the logon procedure. For further information about processing checkout compiler modules, see the manual OS PL/I Checkout Compiler: TSO User's Guide.

The resident library PLICMIX is chosen. The option STEP(5000) is passed to the interpreter stage of the checkout compiler. Notice that the checkout compiler procedure is specified first in the LOADGO command.

```
READY
allocate file(sysplic) dataset('sys1.pliclnk') shr
READY
allocate file(sysut1) block(1024) space(60,60)
READY
allocate file(sysitext) dataset(tomproc.itext)
READY
loadgo (tomproc,'revert') 'step(5000)' plicmix
V1 R3  PL/I CHECKOUT    6 MAR  79          interpretation stage
  TIME 15.33.39                             checkout compiler
OPTIONS SPECIFIED                           invoked
 STEP(5000)

IEN1184A 435X 'FINISH' RAISED.
    AT 920 IN TOP

?go                                       GO subcommand of PLIC command
INTERPRET TIME        0.01 MINS             ends interpretation stage
  TOTAL TIME       0.01 MINS                of checkout compiler
READY
```

COMPILE, LINK-EDIT, AND EXECUTE

The LINK command is used to link-edit a compiled external procedure, or to combine two or more procedures to form a single load module. A load module can be executed by means of the CALL command. You should use the LINK-and-CALL method of processing when you wish to:

- Keep a copy of the load module on an external data set.
- And either link-edit two or more external procedures to form a single program
- Or execute a program repeatedly, without making amendments to the PL/I source code

If you wish to execute a compiled program without keeping a copy of the load module, you can use the LOADGO command (see previous section).

Further information about using the linkage editor is given in the programmer's guide for the optimizing compiler.

Example 1

Two PL/I external procedures are compiled and link-edited to form a single load module, which is then executed. The PL/I source code is held on two data sets, HIBBS.MEDICS.PLI, and HIBBS.MED4.PLI.

The two procedures are compiled using the PLI command. No data set names are specified for the object modules, so the system generates names from the names of the source data sets. The data sets will be called HIBBS.MEDICS.OBJ and HIBBS.MED4.OBJ.

In the LINK command, the names of the two data sets holding the object modules are specified. No name is specified for the data set that will hold the load module, so the system generates the name HIBBS.MEDICS.LOAD(TEMPNAME), based on MEDICS, the first name in the LINK command.

The simple name MEDICS is specified in the CALL command; the system generates the full name HIBBS.MEDICS.LOAD(TEMPNAME).

```
logon hibbs
HIBBS LOGON IN PROGRESS AT 14:51:27 ON FEBRUARY 7, 1979
READY
pli medics
PL/I OPTIMIZER V1 R3.0 TIME: 14.52.15  DATE 7 FEB 1979


NO MESSAGES PRODUCED FOR THIS COMPILATION
COMPILE TIME     0.02 MINS          SPILL FILE:     0 RECORDS, SIZE  4051
READY
pli med4
PL/I OPTIMIZER V1 R3.0 TIME: 14.54.15 DATE 7 FEB 1979


NO MESSAGES OF SEVERITY W AND ABOVE PRODUCED FOR THIS COMPILATION
MESSAGES SUPPRESSED BY THE FLAG OPTION:  1 I.
COMPILE TIME    0.02 MINS          SPILL FILE:     0 RECORDS, SIZE  4051
READY
link (medics,med4) plibase
READY
call medics
TEMPNAME ASSUMED AS A MEMBER NAME
READY
```

If the main procedure had a parameter which was to be passed the CALL command would take the form:

```
call medics '/kildare'
```

where KILDARE was the parameter. If, in addition, the execution time option NOSPIE was required, the command would take the form:

```
call medics 'nospie/kildare'
```

Example 2

The two external procedures compiled in example 1 are link-edited and executed again, but this time a data set name is specified for the load module. The full name of the data set member will be HIBBS.MED.LOAD(M4). The simple name is specified both in the LOAD operand of the LINK command and in the CALL command.

```
link (medics,med4) load(med(m4)) plibase
READY
call med(m4)
READY
```

Example 3

The object module of the MEDICS external procedure is link-edited with a module translated by the PL/I Checkout Compiler. This module is held on a data set called HIBBS.STD7F.OBJ. Before this module can be executed, the data set holding the corresponding intermediate text must be allocated. This data set is called HIBBS.STD7F.ITEXT. (For further information about processing checkout compiler procedures, see the manual OS PL/I Checkout Compiler: TSO User's Guide.) It is assumed that the data set holding the checkout compiler has been allocated in the logon procedure. Otherwise, it must be allocated before the load module is called, as in Example 3 of the section "Compile, Load, and Go."

Notice that in the LINK command, the data set holding the checkout compiler module is the first one in the list. Also notice that the resident libary PLICMIX has been selected.

The load module will be stored as HIBBS.MED.LOAD(M7F); in other words, it is another member of the same data set as the load module generated in Example 2.

The load module is executed by means of the CALL command.

```
READY
link (std7f,medics) load(med(m7f)) plicmix

READY
allocate file(sysitext) dataset(std7f.itext)
READY
call med(m7f)
V1 R2.0 PL/I CHECKOUT     5 FEB  79          interpretation stage of
  TIME 13.21.45                               checkout compiler
                                              invoked
IEN1184A 1095X 'FINISH' RAISED.               normal end of processing
  AT 4010 IN MED

?go                                           GO subcommand of PLIC command
INTERPRET TIME            0.39 MINS             ends interpreter stage of
  TOTAL TIME          0.39 MINS               checkout compiler
READY
```

USING THE RUN FACILITY

RUN can be used as a command or a subcommand of the EDIT command. In the example below, it is used as a subcommand. A data set is created under the EDIT command and then compiled and executed under the RUN command. The argument 'noxtest' is passed to the main PL/I procedure. After successful execution of the program, the compiler terminates and edit submode of the EDIT command is reentered. If the run had detected any errors, they could have been corrected at this stage using subcommands of EDIT. The source data set is then saved and the EDIT command is ended.

```
edit delin pli new
INPUT
00010 delin:proc (parm) options(main);
00020 dcl parm char(100) var;
       .
       .
       .
00450 end;
00460
EDIT
run '/noxtest' opt
PL/I OPTIMIZER V1 R2.0 TIME: 15.24.10  DATE: 7 FEB 79


NO MESSAGES PRODUCED FOR THIS COMPILATION
COMPILE TIME     0.03 MINS        SPILL FILE:      0 RECORDS, SIZE  4051
EDIT
save
SAVED
end
READY
```

Chapter 4: I/O and Attention Interrupts

INTRODUCTION

When a PL/I program is invoked conversationally, data can be transmitted between it and the following:

- The terminal. You can type and enter data when it is needed by the program, and have output sent to the terminal when it is produced by the program.
- Data sets on direct-access storage. Your program can read input data from existing data sets, and can write output data to existing data sets and to new ones.

In neither case does your program need any special PL/I code. The I/O statements can be exactly the same as for a batch mode job.

There are a few special considerations you need to be aware of for conversational I/O, explained in the sections "Conversational Input" and "Conversational Output."

Data sets used by the compiler during compilation also need to be allocated, although the system will generally do this for you. The section "Compiler Data Sets," describes the data sets required by the compiler.

A further conversational facility is the use of attention interrupts and the ATTENTION condition. This is described at the end of this chapter.

CONVERSATIONAL INPUT

You can enter data at the terminal for an input file in your PL/I program, provided the file has been:

1. Declared explicitly or implicitly with the CONSECUTIVE environment option. All stream files meet this condition.
2. Allocated to the terminal.

The standard default input file SYSIN generally meets both these conditions. It is a stream file, and is allocated to the terminal in the logon procedure PLIXLGN.

You are prompted for input to stream files by a colon(:). Each time a GET statement is executed, the system initiates a skip at the terminal to position one of the next line, displays the colon, and initiates a second skip to position one of the following line. You can then enter the required data. If you enter a line that does not contain enough data to complete execution of the GET statement, a further prompt, consisting of a plus sign followed by a colon (+:), is displayed.

You can delay transmission of the data to your program until two or more lines have been entered by adding a hyphen to the end of any line that is to be continued. The hyphen is a TSO feature, and is known as a continuation character.

You may wish to include, in your program, output statements that prompt you for input. If you do so, you can inhibit the initial system prompt by ending your own prompt with a colon. The GET statement could be preceded by a PUT statement such as:

```
PUT SKIP LIST('ENTER NEXT ITEM:');
```

To inhibit the system prompt for the next GET statement, your own prompt must meet the following conditions.

1. It must be either list-directed or edit-directed, and if list-directed, must be to a PRINT file.

2. The file transmitting the prompt must be allocated to the terminal. The system prompt will not be inhibited if the file is merely being copied at the terminal by means of the MONITOR subcommand.

Format of Data: The data you enter at the terminal should have exactly the same format as stream input data in batch mode, except for the following variations:

- Simplified punctuation for input: If you enter separate items of input on separate lines, there is no need to enter intervening blanks or commas; the compiler will insert a comma at the end of each line. For instance, in response to the statement:

  ```
  GET LIST(I,J,K);
  ```

  your terminal interaction could be as follows:

  ```
  :
  1
  +:2
  +:3
  ```

  with a carriage return following each item. It would be equivalent to:

  ```
  :
  1,2,3
  ```

  If you wish to continue an item onto another line, you must end the first line with a continuation character. Otherwise, for a GET LIST or GET DATA statement, a comma will be inserted, and for a GET EDIT statement, the item will be padded (see next paragraph).

- Automatic padding for GET EDIT: There is no need to enter blanks at the end of a line of input for a GET EDIT statement. The item you enter will be padded to the correct length. For instance, for the PL/I statement:

  ```
  GET EDIT(NAME)(A(15));
  ```

  you could enter the five characters:

  ```
  SMITH
  ```

  followed immediately by a carriage return. The item will be padded with 10 blanks, so that the program receives a string 15 characters long. If you wish to continue an item on a second or subsequent line, you must add a continuation character to the end of every line except the last; the first line transmitted would otherwise be padded and treated as the complete data item.

- SKIP option or format item: A SKIP in a GET statement asks the program to ignore data not yet entered. All uses of SKIP(n) where n is greater than one are taken to mean SKIP(1); SKIP(1) is taken to mean that all unused data on the current line is to be ignored.

Stream and Record Files: Both stream and record files can be allocated to the terminal. However, no prompting is provided for record files, and if more than one file is allocated to the terminal, and one or more is a record file, the files' output will not necessarily be synchronized. There is no guarantee that the order in which data is transmitted to and from the terminal will be the same as the order in which the corresponding PL/I input/output statements are executed. It is therefore advisable to use stream files wherever possible.

Capital and Lowercase Letters: Character-strings are transmitted to the program as entered. Assuming that your terminal has both sets of characters, you must type in lowercase or capital letters, according to the requirements of your program. There is no translation of the input string from lowercase to capital letters or vice versa. If the string is to be compared with a character-string constant in the program, remember that if the program was created using the EDIT command without the ASIS operand, the constant will be in capitals whether it was entered in lowercase or capitals.

End of File: The characters /* in positions one and two of a line that contains no other characters are treated as an end-of-file mark, that is, they raise the ENDFILE condition.

COPY Option of GET Statement: The GET statement can specify the COPY option, but if the COPY file, as well as the input file, is allocated to the terminal, no copy of the data will be printed. Note that SYSPRINT is the default COPY file, and that this file is allocated to the terminal by default in the standard LOGON procedure PLIXLGN.

CONVERSATIONAL OUTPUT

You can obtain at your terminal data from a PL/I file that has been both:

1. Declared explicitly or implicitly with the CONSECUTIVE environment option. All stream files meet this condition.

2. Allocated to the terminal.

The standard print file SYSPRINT generally meets both these conditions. It is a stream file, and is allocated to the terminal in the standard logon procedure PLIXLGN.

Format of PRINT Files: Data from SYSPRINT or other PRINT files is not normally formatted into pages at the terminal. Three lines are always skipped for PAGE and LINE options and format items. The ENDPAGE condition is normally never raised. SKIP(n), where n is greater than three, causes only three lines to be skipped. SKIP(0) is implemented by backspacing, and should therefore not be used with terminals that do not have a backspace feature, such as the IBM 2260 and 2265.

You can cause a PRINT file to be formatted into pages by inserting a tab control table in your program. The table must be called PLITABS, and its contents are explained in the programmer's guide for the optimizing compiler. The element PAGELENGTH must be initialized to the length of page you require, that is, the length of the sheet of paper on

which each page is to be printed, expressed as the maximum number of lines that could be printed on it. The element PAGESIZE must be initialized to the actual number of lines to be printed on each page. After the number of lines in PAGESIZE has been printed on a page, ENDPAGE is raised, for which standard system action is to skip PAGELENGTH minus PAGESIZE lines, and then start printing the next page. For otherwise standard layout, the other elements in PLITABS must be initialized to the values shown in the programmer's guide. You can also use PLITABS to alter the tabulating positions of list-directed and data-directed output.

Although some types of terminal have a tabulating facility, tabulating of list-directed and data-directed output is always achieved by transmission of blank characters.

Stream and Record Files: Both stream and record files can be allocated to the terminal. However, if more than one file is allocated to the terminal and one or more is a record file, the files' output will not necessarily be synchronized. There is no guarantee that the order in which data is transmitted between the program and the terminal will be the same as the order in which the corresponding PL/I input and output statements are executed. In addition, because of a TCAM restriction, any output to record files at the terminal is printed in uppercase (capital) letters. It is therefore advisable to use stream files wherever possible.

Capital and Lowercase Characters: For stream files, characters are displayed at the terminal as they are held in the program, provided the terminal can display them. For instance, with an IBM 2741 terminal, capital and lowercase letters are printed as such, without translation. For record files, all characters are translated to uppercase. A variable or constant in the program can contain lowercase letters if the program was created under the EDIT command with the ASIS operand, or if the program has read lowercase letters from the terminal.

COMPILER DATA SETS

The compiler requires the use of a number of data sets in order to process a PL/I program. These are listed in Figure 3. The following data sets are always required by the compiler.

- The data set holding the PL/I program
- A data set for the compiler listing

Up to six more data sets may be required, depending on which compiler options have been specified.

The data sets must be allocated before the compiler uses them. If the PLI command or the RUN command or subcommand is used, the compiler will be invoked via the prompter, and the prompter will allocate the necessary data sets. If you invoke the compiler without the prompter, you must allocate the necessary data sets yourself.

When the prompter allocates compiler data sets, it uses ddnames generated by TSO rather than the ddnames that are used in batch mode. Figure 3 includes the batch-mode ddnames of the data sets, but their main purpose here is to help you identify the data sets. If the compiler is invoked via the prompter, you cannot refer to the data sets by these names. To control the allocation of compiler data sets, you need to use the appropriate operand of the PLI command. For instance, to allocate the standard output file (ddname SYSPRINT in batch mode) to the terminal, you should use the PRINT(*) operand of the PLI command; you cannot make the allocation by using the ALLOCATE command with FILE(SYSPRINT) and DATASET(*) operands. Figure 3 shows which operands to use for those data sets whose allocation you can control.

When the prompter is not invoked, the batch-mode ddnames will be recognized as referring to the compiler data sets.

Data set (and batch-mode ddname)	When required	Where to specify data set in PLI command	Descriptive qualifier	Allocated by	Parameters used by prompter[1] SPACE=[2]	DISP=[3]
Primary input (SYSCIN or SYSIN)	Always	1st operand	PLI	Prompter	-[4]	SHR
Temporary work data set(SYSUT1)	When insufficient main storage	Cannot specify	-	Prompter	(1024,(60,60))	(NEW,DELETE)
Compiler listing (SYSPRINT)	Always	Argument of PRINT operand	LIST	Prompter	(629,(n,m))	(OLD,KEEP) or[5] (NEW,CATLG)
Object module (SYSLIN)	When OBJECT option applies	1st argument of OBJECT operand	OBJ	Prompter, when required[7]	(400,(50,50))	(OLD,KEEP)) or (NEW,CATLG)
Object module or preprocessor output in card format (SYSPUNCH)	When MACRO and MDECK options apply	Argument of MDECK DECK operand	DECK	Prompter, when required[7]	(400,(50,50))	(OLD,KEEP) or (NEW,CATLG)
Secondary input to preprocessor (SYSLIB)[6]	When LIB operand used	Arguments of LIB operand	-	Prompter, when required	-[6]	SHR

[1] Unit is determined by entry in User Attribute Data Set.

[2] These space allocations apply only if the data set is new. The first argument of the SPACE parameter establishes the block size. For the SYSUT1, SYSPRINT, SYSLIN, and SYSPUNCH data sets, the record format, record length, and number of buffers are established by the compiler when it opens the data sets. The values it uses are given in the compiler's programmer's guide.

[3] The prompter first tries to allocate the SYSPRINT, SYSLIN, and SYSPUNCH data sets with DISP=(OLD,KEEP). This will cause any existing data set (or partitioned data set member) with the same name to be replaced with the new one. If the data set name cannot be found in the system catalog, the data set is allocated with DISP=(NEW,CATLG).

[4] The data set already exists, therefore SPACE (and also UNIT) are already established.

[5] DISP parameter used only if PRINT(dsname) operand applies. Otherwise, prompter supplies following parameters:
TERM=TS if PRINT(*) operand applies
DUMMY if NOPRINT operand applies
SYSOUT if SYSPRINT operand applies

[6] If any ddnames are specified in %INCLUDE statements, allocate the data sets with the ALLOCATE statement.

[7] Except when the associated option has been specified by means of a *PROCESS statement. In this case the data set(s) must be allocated by the user.

Figure 3. Compiler data sets.

EXAMPLE OF INPUT/OUTPUT

The example program prints a report based on information retrieved from a data base. The content of the report is controlled by a list of parameters that contains the name of the person requiring the report and a set of numbers indicating the information that is to be printed. In the example, the parameters are read from the terminal. The program includes a prompt for the name parameter, and a message confirming its acceptance. The report is printed on a system output device. The program uses four files:

SYSPRINT	Standard stream output file. Prints prompt and confirmation at terminal.
PARMS	Stream input file. Reads parameters from terminal.
INBASE	Record input file. Reads data base, namely, member MEM3 of data set BDATA.
REPORT	Sends report to SYSOUT device.

SYSPRINT has been allocated to the terminal by the logon procedure. The other three files are allocated by ALLOCATE commands entered in TSO submode.

The program is called REPORTR and it is held on a conventionally-named TSO data set whose user-supplied name is REPORTER. The compiler is invoked with the SOURCE option to provide a list of the PL/I source code.

```
READY
pli reporter print(*) source                "print(*)" allocates
OPTIMIZING COMPILER INVOKED                   source listing to terminal
PL/I OPTIMIZER V1 R2.0 TIME:10.48.34  DATE: 17 MAR 79
OPTIONS SPECIFIED
 SOURCE


                    SOURCE LISTING
 NUMBER
      10 00000010 REPORTR:PROC OPTIONS(MAIN);
              .
              .
     180 00000180 ON ENDFILE(PARMS) GO TO READER;
              .
              .
    1000 00001000 PUT LIST('ENTER NAME:');  print prompt at terminal
    1010 00001010 GET FILE(PARMS) LIST(NAME);
              .                             read name parameter from
              .                               terminal
    1050 00001050 PUT LIST('NAME ACCEPTED');
              .                             confirmation message
              .
    2000 00002000 GET FILE(PARMS) LIST((A(I) DO I=1 TO 50));
              .                             read other parameters from
              .                               terminal
    2010 00002010 READER:
         00002020 READ FILE(INBASE) INTO(B);
              .                             read data base
              .
    4010 00004010 PRINTER:
         00004020 PUT FILE(REPORT) EDIT(HEAD1||NAME);
              .                             print line of report on
              .                               system printer
    5000 00005000 END REPORTR;
```

```
NO MESSAGES PRODUCED FOR THIS COMPILATION
COMPILE TIME     0.30 MINS        SPILL FILE:      0 RECORDS, SIZE 3491

READY
alloc file(parms) dataset(*)                    file to read parameters from
READY                                             terminal
alloc file(inbase) dataset('bdata(mem3)') old
READY                                           file to read data base
alloc file(report) sysout                       file to print report on
READY                                             system printer
loadgo reporter plibase

ENTER NAME: 'F W WIlliams'                      prompt & name parameter
NAME ACCEPTED                                   confirmation message
:                                               automatic prompt for params.
1 3 5 7 10 14 15 19                             parameters entered
+:/*                                            prompt for further params.
READY                                             End-of-file entered
```

ATTENTION INTERRUPTS

The INTERRUPT option allows attention interrupts to become an integral part of programming with the optimizing compiler, and this gives the user considerable interactive control of the program.

If the INTERRUPT option is in effect during compilation, the compiled program will respond to one attention interrupt by searching for an established ATTENTION on-unit, executing it if it finds one, and continuing with the processing if it does not. When the execution of an ATTENTION on-unit is complete, control will return to the point of interrupt unless directed elsewhere by means of a GOTO statement. Two attention interrupts terminate execution.

If NOINTERRUPT was in effect during compilation, the compiled program will be terminated if one attention interrupt occurs.

Attention interrupts can be simulated with any type of terminal. The TERMINAL command is used for this purpose and described in Appendix B, "Using a Terminal other than a 2741."

With a 2741 terminal, the simplest way to cause an attention interrupt is to press the ATTN key. If the terminal is fitted with a Receive Interrupt special feature, the system will respond to the ATTN key at any time. If the terminal does not have this feature, you can initiate an attention interrupt only when the system is waiting for a command, subcommand, or data.

How to Cause an Attention Interrupt

One attention interrupt is caused on a 2741 by pressing the attention button once. Two are caused by pressing the attention button twice in quick succession.

HOW TO USE ATTENTION INTERRUPTS

The ability given by the INTERRUPT option to respond or not respond to attention interrupts allows for two possible uses:

1. Attention interrupts can be used purely as a debugging feature with ATTENTION on units used to supply debugging data. The program can finally be compiled with NOINTERRUPT for production purposes.

2. Production programs which are run from a terminal can be made more responsive and convenient to use by the introduction of ON ATTENTION on-units.

Attention on-units used for debugging

When debugging under the optimizing compiler, ATTENTION on-units can be used to transmit values to the terminal when an attention interrupt is caused. For example, an ATTENTION on-unit might read:

```
ON ATTENTION PUT DATA(A,B,C,ICOUNT);
```

These values would then be transmitted to the terminal when an attention interrupt was caused. When the program had been debugged, the unit could be retained and the program compiled with the NOINTERRUPT option. This would prevent code designed to poll for attention interrupt being included in the load module and so there would be no time overhead. There would, however, be a small space overhead because the on-unit itself would be compiled.

The use of NOINTERRUPT also allows programs compiled on the checkout compiler with debugging ATTENTION on-units to be compiled on the optimizing compiler without producing an execution time overhead.

Attention on-units used for Interactive Systems

Full ON ATTENTION support by the optimizing compiler also improves the convenience of using conversational programs in a production environment, by allowing the user to interrupt unwanted processing. For example, the user may wish to respecify the criteria for a data base inquiry without waiting for the entire output to be displayed.

Typically, the ATTENTION on-unit will prompt the user for input and carry out some action determined by that input. For example:

```
   ON ATTENTION BEGIN;
     ERRCOUNT=0;
     PUT EDIT
     ('ENTER 1 FOR NEXT TABLE, 2 FOR REPETITION OF CURRENT TABLE 3
        TO END OUTPUT:')(A);
FIRST:
     GET EDIT (NUM)(A(1));
```

```
      SELECT(NUM);
        WHEN(1) GOTO NEXT;
        WHEN(2) GOTO START;
        WHEN(3) GOTO FINAL;
      OTHERWISE DO;
        ERRCOUNT=ERRCOUNT+1;
        IF ERRCOUNT<3 THEN  DO;
           PUT EDIT('INCORRECT INPUT TRY AGAIN')(A);
           GOTO FIRST;
           END;
        ELSE SIGNAL ERROR;
      END;  /*OTHERWISE CLAUSE*/
    END;  /*SELECT*/
  END;  /*ON-UNIT*/
```

The terminal interaction resulting from causing an attention interrupt could be as follows:

```
call bicent
THIS PROGRAM LISTS TABLES OF DATA RELATING
TO AMERICAN BICENTENNIAL CELEBRATIONS
USE ATTENTION INTERRUPT TO CHANGE TABLE
LIST OF STATE BIRDS

AMERICAN ROBIN                  CONNECTICUT, MICHIGAN, WISCONSIN
BALTIMORE ORIOLE                MARYLAND
BLACK CAPPED CHICKADEE          MAINE,MASSACHUSETS
(Attention interrupt here)

ENTER 1 FOR NEXT TABLE, 2 FOR REPEAT OF CURRENT TABLE, 3 TO END OUTPUT:1

LIST OF STATE FLOWERS
AMERICAN BEAUTY ROSE            DISTRICT OF COLUMBIA
APPLE BLOSSOM                   ARKANSAS,MICHIGAN
ARBUTUS                         MASSACHUSETTS
etc
```

Background to Attention Handling

If you are going to make extensive use of attention interrupts, it is important to understand something of how they are implemented by the optimizing compiler.

Essentially, causing an attention interrupt sets a switch immediately and this switch is tested by means of polling at suitable points in the compiled program.

In procedures compiled with the INTERRUPT option, polling takes place between PL/I statements at branch-in points. Polling also takes place in all stream I/O statements to and from the terminal if any procedure in the load module was compiled with the INTERRUPT option. This arrangement allows maximum control of terminal input and output with minimum performance overheads. It also ensures that the ATTENTION condition is raised between PL/I statements.

Pitfalls when using Attention Interrupts

The synchronization of terminal printout and processing by the CPU and the method used of implementing the ATTENTION condition cause various pitfalls for the user of attention interrupts. These are described below.

Synchronization

When output is being transmitted to the terminal, the statement being executed in the CPU may be well beyond the point where the output is transmitted. (The number of buffers allocated during TSO installation determines how far.) Consequently, an attention interrupt will often cause loss of output that is held in buffers. In addition, an attempt to end excessive output to the terminal by use of an attention interrupt may have unexpected results if the program is not actually executing the output statement when the attention interrupt is caused.

Consider the on-unit

```
     ON ATTN BEGIN;
     /*Unit illustrates a potential pitfall*/
         ON ATTN GOTO ENDUNIT; /*Second ON statement kills the
                                 output if too long, by accepting
                                 attention interrupt during output*/
         PUT DATA;
ENDUNIT:END;
```

An attention interrupt entered when you have seen enough output may in fact occur when the unit has completed executing. Thus the attention, far from ending the output, will just cause another set of output to begin.

Synchronization is only carried out when a GET statement to the terminal is executed. Therefore, a GET statement at the end of the unit would solve the problem. A corrected on-unit could read:

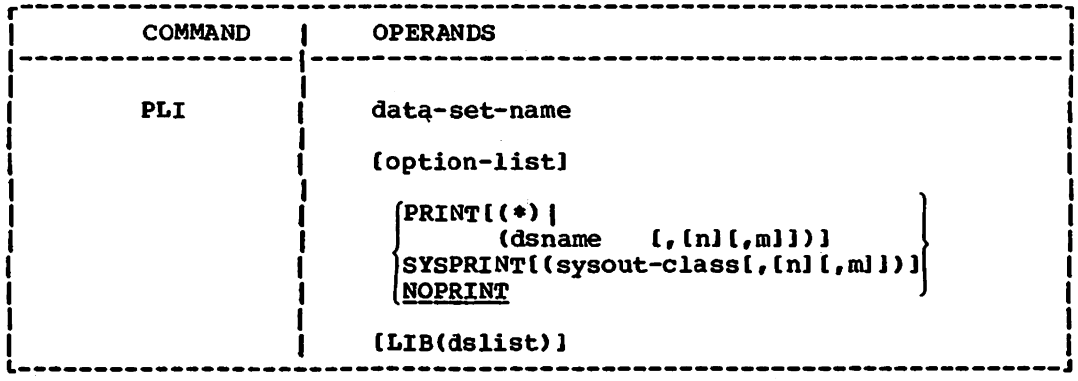
```
     ON ATTN BEGIN;
      ON ATTN GOTO ENDUNIT;
      PUT EDIT
('TO END OUTPUT CAUSE ATTENTION. THEN ENTER GO TO CONTINUE OR STOP TO STOP')(A);
      DCL ANS CHAR(4) INIT('');
      PUT DATA;
 ENDUNIT:
/*Execution will wait here to synchronize the GET statement. Therefore,
attention interrupts entered during output of data will occur within the
scope of the on-unit, so data will be ended by second ON ATTN
statement*/
      GET EDIT (ANS)(A(4));
      UNSPEC(ANS)=UNSPEC(ANS)|(4) '01000000'B;
         /*Fold to upper case because input may be in upper or lower*/
      IF ANS='STOP' THEN STOP;
      END;
```

Note that the prompt for the GET statement must appear before the PUT DATA or it will be lost when an attention interrupt occurs.

Programs Partly Compiled with the INTERRUPT Option

If any procedures within a load module have been compiled with the INTERRUPT option, a STAX macro instruction is issued at the start of execution. Consequently, an attention interrupt will be noted whenever it is caused. The ATTENTION condition itself will not be raised until later, since PL/I may be in the process of constructing control blocks at the time the attention interrupt is noted. It will be raised during stream I/O to or from the terminal in all procedures and at branch-in points in procedures compiled with the INTERRUPT option. If you wish to use attention interrupts for debugging purposes, the results may be unexpected because any attention on units will be executed regardless of the option with which the procedure that contains them was compiled.

Use the PLI command to compile a PL/I program. The command invokes the PL/I Optimizing Compiler.



data-set-name
specifies the name of the primary input data set for the optimizing compiler. This can be either a fully qualified name (enclosed in single quotation marks) or a simple name (for which the prompter will add the identification qualifier, and the descriptive qualifier PLI). This must be the first operand specified.

option-list
specifies one or more compiler options which are to apply for this compilation.

The compiler options that can be specified in a TSO environment are described in the following section. Programmers familiar with batch processing should note that defaults are altered for TSO, and that the DECK, MDECK, and OBJECT options have been extended to allow names of data sets onto which the output will be written to be specified.

Separate the options by at least one blank or one comma; you can add any number of extra blanks. The order of the options is unimportant. In fact, the PRINT/NOPRINT and LIB operands can be interspersed in the option-list; they are recognized by their keywords. If two contradictory options are specified, the last is accepted and the first ignored.

Options specified in the PLI command can be subsequently overridden by options specified on *PROCESS compiler control statements in the primary input. If the DECK, MACRO, MDECK, and OBJECT options are required for any program in a batched compilation the option should be specified in the PLI command so that the prompter allocates the required data sets. The negative forms can then be used on the PROCESS statements for the programs that do not require the option.

PRINT(*)
specifies that the compiler listing, on the SYSPRINT file, is

to be written at the terminal; no other copy will be available.

PRINT(dsname[,[n][,m]])

specifies that the compiler listing, on the SYSPRINT file, is to be written on the data set named in parentheses. This can be either a fully qualified name (enclosed in single quotation marks) or a simple name (for which the prompter will add the identification qualifier, and the descriptive qualifier LIST).

If no dsname argument is specified for the PRINT operand, the prompter will add the identification and descriptive qualifiers to the data set name specified in the first operand, producing a data set name of the form 'user-identification.user-supplied-name.LIST'. If "dsname" is not specified and the first operand of PLI specifies a member of a partitioned data set, the member name will be ignored - the generated data set name will be based on the name of the partitioned data set.

n,m Specifies the space allocation in lines for the listing dataset. It should be used when the size of the listing has caused a B37 abend during compilation.

- n specifies the number of lines in the primary allocation.
- m specifies the number of lines in the secondary allocation.

If n is omitted the preceding comma must be included. For example to enter only the size of the secondary allocation and accept the default for the primary you would enter: SYSPRINT(PRINTDS,,500).

The space allocation used if n and m are not specified is that specified during compiler installation.

SYSPRINT[(sysout-class[,n[,m]])]

specifies that the compiler listing, on the SYSPRINT file, is to be written to the sysout class named in parentheses. If no class is specified, the output is written to a default sysout class; the IBM standard for this default is class A. For meaning of n, m see "PRINT" above.

NOPRINT

specifies that the compiler listing is not to be produced on the SYSPRINT file. You can still get most of the listing written at the terminal by using the TERMINAL compiler option.

LIB(dslist)

specifies one or more data sets that are to be used as the secondary input to the preprocessor. These data sets will be concatenated in the order specified and then associated with the ddname SYSLIB. This will override any previous ALLOCATE statement with a FILE(SYSLIB) operand. If you have specified any other ddname in the %INCLUDE statements in the PL/I program, you must allocate the data sets associated with that ddname yourself.

The data set names can be either fully qualified (each enclosed in single quotation marks) or simple names (for which the prompter will add the identification qualifier, but no descriptive qualifier).

Separate the data set names by at least one blank or one comma; you can add any number of extra blanks.

If you use the LIB operand, either the INCLUDE or the MACRO compiler option must also apply.

Example 1

Operation: Invoke the PL/I optimizing compiler to process a PL/I program.

Known: User-identification is ABC
Data set containing the program is named ABC.UPDATE.PLI
SYSPRINT file is to be directed to the terminal.
Default options and data set names are to be used.

```
PLI  UPDATE  PRINT(*)
```

Example 2

Operation: Invoke the PL/I optimizing compiler to process a PL/I program.

Known: User-identification is XYZ.
Data set containing the program is named ABC.MATRIX.PLI
SYSPRINT file is to be written on a data set named MATLIST.
MACRO and MDECK options are required, with the associated output to be written on a data set named MATCARD.
Secondary input to preprocessor to be read from a library named XYZ.SOURCE.
Otherwise default options and data set names are to be used.

```
PLI  'ABC.MATRIX.PLI'  PRINT('MATLIST'),MACRO,MDECK('MATCARD'),LIB(SOURCE)
```

Chapter 6: Compiler Options

This chapter describes the options that can be specified for the PL/I optimizing compiler in a TSO environment. The keywords, abbreviations, and IBM defaults are listed in Figure 4; the functions of the options are shown in Figure 5. Most options comprise positive and negative forms, the latter beginning with NO. Where negative forms exist, one or other form is applied by default. Where no negative forms exist, the option must in most cases, be explicitly specified.

Note: Your installation may have adopted a different set of defaults from the standard IBM defaults shown here.

The abbreviations listed in Figure 4 are the standard abbreviations recognized by the optimizing compiler. In addition, the PL/I prompter will accept the following standard TSO forms:

LINECNT	for	LINECOUNT
LOAD	for	OBJECT
NOLOAD	for	NOOBJECT
CHAR60	for	CHARSET(60)
CHAR48	for	CHARSET(48)
EBCDIC	for	CHARSET(EBCDIC)
BCD	for	CHARSET(BCD)

Keywords can be shortened by deleting any number of characters on the right provided the result remains non-ambiguous. As the prompter is also used to invoke the PL/I checkout compiler, you must take into account the additional options which apply only to the checkout compiler (that is, BLOCK, COMPATIBLE, DIAGNOSE, FORMAT, HALT, ISASIZE, RUN, SMAN, STEP, and STEPLINES) when deciding whether an abbreviation is non-ambiguous.

Each option is considered to be a separate operand and must follow TSO syntax conventions. Thus, in arguments containing two or more items, items can be omitted only from the end of the list (for instance, MARGINS(2,70) would be accepted, while MARGINS(,70) would not). The items must be separated by one blank or one comma; though any number of additional blanks may be added.

Several of the options cause tables, etc., to be included in the compiler listing that is written on the standard output file. A full description of this listing is given in the programmer's guide for the optimizing compiler. If you wish to have these listings directed to the terminal, you can specify them in the TERMINAL option (see TERMINAL in the following list). Alternatively, you can specify a PRINT(*) operand in the PLI command, in addition to the list of options.

Compiler Option Keywords	Abbreviations	IBM Default for TSO Environment
AGGREGATE|NOAGGREGATE	AG|NAG	NOAGGREGATE
ATTRIBUTES[(FULL|SHORT)]| NOATTRIBUTES	A[(F|S)]|NA	NOATTRIBUTES (FULL[1])
CHARSET (48|60,EBCDIC|BCD)	CS(48|60,EB|B)	CHARSET (60,EBCDIC)
COMPILE|NOCOMPILE[(W|E|S)]	C|NC[(W|E|S)]	NOCOMPILE(S)
CONTROL('password')	-	-
COUNT|NOCOUNT	-	NOCOUNT
DECK|NODECK	D|ND	NODECK
DUMP|NODUMP	DU|NDU	NODUMP
ESD|NOESD	-	NOESD
FLAG[(I|W|E|S)]	F[(I|W|E|S)]	FLAG(W)
FLOW[(n,m)]|NOFLOW	-	NOFLOW
GONUMBER|NOGONUMBER	GN|NGN	NOGONUMBER
GOSTMT|NOGOSTMT	GS|NGS	NOGOSTMT
IMPRECISE|NOIMPRECISE	IMP|NIMP	NOIMPRECISE
INCLUDE|NOINCLUDE	-	NOINCLUDE
INSOURCE|NOINSOURCE	IS|NIS	INSOURCE
INTERRUPT|NOINTERRUPT	INT|NINT	NOINTERRUPT
LINECOUNT(n)	LC(n)	LINECOUNT(55)
LIST(n,m)|NOLIST	-	NOLIST
LMESSAGE|SMESSAGE	LMSG|SMSG	SMESSAGE
MACRO|NOMACRO	M|NM	NOMACRO
MAP|NOMAP	-	NOMAP
MARGINI('c')|NOMARGINI	MI('c')|NMI	NOMARGINI
MARGINS(m,n,c)	MAR(m,n,c)	F-format: MARGINS(2,72,0) V-format: MARGINS(10,100,0)
MDECK|NOMDECK	MD|NMD	NOMDECK
NAME('name')	N('name')	-
NEST|NONEST	-	NONEST
NUMBER|NONUMBER	NUM|NNUM	NUMBER
OBJECT|NOOBJECT	OBJ|NOBJ	OBJECT
OFFSET|NOOFFSET	OF|NOF	NOOFFSET
OPTIMIZE(TIME|0|2)| NOOPTIMIZE	OPT(TIME|0|2)| NOPT	NOOPTIMIZE
OPTIONS|NOOPTIONS	OP|NOP	NOOPTIONS
SEQUENCE(m,n)	SEQ(m,n)	F-format: SEQUENCE(73,80) V-format: SEQUENCE(1,8)
SIZE(yyyyyy|yyyK|MAX)	SZ(yyyyyy|yyyK|MAX)	SIZE(MAX)
SOURCE|NOSOURCE	S|NS	NOSOURCE
STMT|NOSTMT	-	NOSTMT
STORAGE|NOSTORAGE	STG|NSTG	NOSTORAGE
SYNTAX|NOSYNTAX[(W|E|S)]	SYN|NSYN[(W|E|S)]	NOSYNTAX(S)
TERMINAL[(option-list)]| NOTERMINAL	TERM[(option-list)]| NTERM	TERMINAL
XREF[(FULL|SHORT)]|NOXREF	X[(F|S)]|NX	NOXREF(FULL[1])

[1]FULL is the default suboption

Figure 4. Compiler option keywords, abbreviations, and defaults

LISTING OPTIONS		
Control listings produced	AGGREGATE	list of aggregates and their size
	ATTRIBUTES[(SHORT|FULL)]	list of attributes of identifiers
	ESD	list of external symbol dictionary
	INSOURCE	list of preprocessor input
	FLAG[(I|W|E|S)]	suppress diagnostic messages below a certain severity
	LIST	list compiled code produced by compiler
	MAP	lists offsets of variables in static control section and DSAs
	OPTIONS	list of options used
	SOURCE	list of source program or preprocessor output
	STORAGE	list of storage used
	XREF[(SHORT|FULL)]	list of statements in which each identifier is used
Improve readability of source listing		
	NEST	indicates do-group and block level by numbering in margin
	MARGINI	highlights any source outside margins
Control lines per page of listing		
	LINECOUNT	specifies number of lines per page on listing
INPUT OPTIONS		
	CHARSET	identify the character set used in source
		identify position of a carriage control character
	SEQUENCE	specify the columns used for sequence numbers
OPTIONS TO PREVENT UNNECESSARY PROCESSING		
	NOSYNTAX[(W|E|S)]	stop processing after errors are found in preprocessing
	NOCOMPILE[(W|E|S)]	stop processing after errors are found in syntax checking

Figure 5. (Part 1 of 2). Compiler options arranged by function

OPTIONS FOR PREPROCESSING		
	INCLUDE	allows secondary input to be included without using preprocessor
	MACRO	allows preprocessor to be used
	MDECK	produces a source deck from preprocessor output
OPTIONS TO USE WHEN PRODUCING AN OBJECT MODULE		
	OBJECT	produce an object module from compiled output
	NAME	specify the name of the object module produced
	DECK	produce an object module in punched card format
OPTIONS TO CONTROL STORAGE USED		
	SIZE	controls the amount of storage used by the compiler
OPTIONS TO IMPROVE USABILITY AT A TERMINAL		
	TERMINAL	specifies how much of listing is transmitted to terminal
	LMESSAGE/SMESSAGE	specifies concise or full message format
OPTIONS TO SPECIFY STATEMENT NUMBERING SYSTEM USED		
	NUMBER & GONUMBER	numbers statements according to line which they start
	STMT & GOSTMT	numbers statements sequentially
	OFFSET	specifies that a listing associating statement numbers with offsets will be generated
OPTIONS FOR USE WHEN DEBUGGING		
	COUNT	count number of times each statement is executed
	FLOW	generate code that will result in a trace of statements executed being retained
OPTIONS TO CONTROL EFFECT OF ATTENTION INTERRUPTS		
	INTERRUPT	specifies that the ATTENTION condition will be raised when interrupt is caused
	NOINTERRUPT	specifies that the use of the attention interrupt will terminate the program
OPTION FOR USE WHEN DEBUGGING COMPILER		
	DUMP	produces a dump if the compiler terminates abnormally (ignored if used in *PROCESS statement)

Figure 5. (Part 2 of 2). Compiler options arranged by function

AGGREGATE

The AGGREGATE option specifies that the compiler is to include in the compiler listing an aggregate length table, giving the lengths of all arrays and major structures in the source program.

ATTRIBUTES[(FULL|SHORT)]|NOATTRIBUTES

IBM default NOATTRIBUTES
IBM default suboption FULL

The ATTRIBUTES option specifies that the compiler is to include in the compiler listing a table of source-program identifiers and their attributes. If both ATTRIBUTES and XREF apply, the two tables are combined.

If SHORT is specified, unreferenced identifiers are omitted, making the listing more manageable.

If both ATTRIBUTES and XREF apply, and there is a conflict between SHORT and FULL, the usage is determined by the last option found. For example ATTRIBUTES(SHORT) XREF (FULL) results in FULL applying to the combined listing.

The default FULL means that FULL applies if the option is specified with no sub-option.

CHARSET(48|60,EBCDIC|BCD)

IBM default CHARSER(60,EBCDIC)

The CHARSET option specifies the character set and data code that you have used to create the source program. The compiler will accept source programs written in the 60-character set or the 48-character set, and in the Extended Binary Coded Decimal Interchange Code (EBCDIC) or Binary Coded Decimal (BCD). It is unlikely you will ever need to use BCD. The 48-character set may be required for terminals with a limited character set.

60- or 48-Character Set: If the source program is written in the 60-character set, specify CHARSET(60); if it is written in the 48-character set, specify CHARSET(48). The language reference manual for this compiler lists both of these character sets. (The compiler will accept source programs written in either character set if CHARSET(48) is specified, however, if the reserved keywords CAT, LE, etc., are used as identifiers, errors may occur.)

BCD or EBCDIC: If the source program is written in BCD, specify CHARSET(BCD); if it is written in EBCDIC, specify CHARSET(EBCDIC). The language reference manual for this compiler lists the EBCDIC representation of both the 48-character set and the 60-character set.

If both arguments are specified, they may be in any order.

COMPILE|NOCOMPILE[(W|E|S)]

IBM default NOCOMPILE(S)

The COMPILE option specifies that the compiler is to compile the source program unless an unrecoverable error was detected during preprocessing or syntax checking. The NOCOMPILE option without an argument causes

processing to stop unconditionally after syntax checking. With an argument, continuation depends on the severity or errors detected so far, as follows:

NOCOMPILE(W)	No compilation if a warning, error, severe error, or unrecoverable error is detected.
NOCOMPILE(E)	No compilation if an error, severe error, or unrecoverable error is detected.
NOCOMPILE(S)	No compilation if a severe error or unrecoverable error is detected.

CONTROL('password')

IBM default: CONTROL does not apply unless specified

The CONTROL option specifies that any compiler options deleted for your installation are to be available for this compilation. You must still specify the appropriate keywords to use the options. The CONTROL option must be specified with a password that is established for each installation; use of an incorrect password will cause processing to be terminated. The CONTROL option, if used, must be specified first in the list of options.

It has the format:

CONTROL('password')

where "password" is a character string, not exceeding eight characters.

COUNT|NOCOUNT

IBM default NOCOUNT

The COUNT option specifies that a list of how many times each statement has been executed is to be produced when the program terminates. The list is written is the PLIDUMP data set, if such a data set has been allocated, otherwise on SYSPRINT. COUNT implies GONUMBER (if NUMBER applies) or GOSTMT (if STMT applies).

DECK[(dsname)]|NODECK

IBM default NODECK

The DECK option specifies that the compiler is to produce an object module in the form of 80-column card images and store it on a data set. Columns 73-76 of each card contain a code to identify the object module; this code comprises the first four characters of the first label in the external procedure represented by the object module. Columns 77-80 contain a 4-digit decimal number: the first card is numbered 0001, the second 0002, etc.

The data set onto which the object module is written will have the name specified in "dsname." This can be a fully qualified name (enclosed in single quotation marks) or a simple name (to which the user identification and the descriptive qualifier DECK will be added). If "dsname" is not specified, the user-supplied name will be taken from the first operand of the PLI command, and the user-identification and descriptive qualifier DECK will be added. If "dsname" is not specified and the first operand of PLI specifies a member of a partitioned data set, the member name will be ignored - the generated data set name will be based on the name of the partitioned data set.

DUMP|NODUMP

IBM default NODUMP

The DUMP option specifies that the compiler is to produce a formatted dump of the contents of the registers and main storage used by the optimizing compiler if compilation terminates abnormally (usually due to an I/O error or compiler error). This dump is written on the data set associated with SYSPRINT. Implementation of the DUMP option necessitates the use of a considerable amount of main storage by routines that handle the dumping. This reduces the storage available for compilation which reduces the speed of the compilation.

ESD|NOESD

IBM default NOESD

The ESD option specifies that the external symbol dictionary (ESD) is to be listed in the compiler listing.

FLAG(I|W|E|S)

IBM default FLAG(W)

The FLAG option specifies the minimum severity of error that requires a message to be listed in the compiler listing and at the terminal. The possible forms of the FLAG option are:

FLAG(I)	List all messages.
FLAG(W)	List all except informatory messages. If you specify FLAG, FLAG(I) is assumed.
FLAG(E)	List all except warning and informatory messages.
FLAG(S)	List only severe error and unrecoverable error messages.

FLOW[(n,m)]|NOFLOW

IBM default NOFLOW

The FLOW option specifies that the compiler is to list the transfers of control most recently executed in the PL/I program prior to the occurrence of an interrupt that results in an execution-time message. The format of the FLOW option is:

```
FLOW[(n,m)]
```

where:

"n" is the maximum number of entries to be included in the list. It should not exceed 32768.

"m" is the maximum number of changes of block to be recorded in the list ("block" here meaning procedure or on-unit). It should not exceed 32768.

"n" and "m" may be omitted; in this case, the standard default values are, respectively, 25 and 10.

The list will start at the earliest information and continue to the latest in order of execution.

GONUMBER|NOGONUMBER

IBM default NOGONUMBER

The GONUMBER option specifies that the compiler is to produce additional information that will allow line numbers from the source program to be included in execution-time messages.

Alternatively, these line numbers can be derived by using the offset address, which is always included in execution-time messages, and the table produced by the OFFSET option. (The NUMBER option must also apply.)

Use of the GONUMBER option implies NUMBER and NOSTMT.

GOSTMT|NOGOSTMT

IBM default NOGOSTMT

The GOSTMT option specifies that the compiler is to produce additional information that will allow statement numbers from the source program to be included in execution-time messages.

Alternatively, these statement numbers can be derived by using the offset address, which is always included in execution-time messages, and the table produced by the OFFSET option. (The STMT option must also apply.)

Use of the GOSTMT option implies STMT and NONUMBER.

IMPRECISE|NOIMPRECISE

IBM default NOIMPRECISE

The IMPRECISE option specifies that the compiler is to include extra text in the object module to localize imprecise interrupts when executing the program with an IBM System/360 Model 91 or 195. This extra text ensures that if interrupts occur, the correct on-units will be entered, and that the correct line or statement numbers will appear in execution-time messages.

INCLUDE|NOINCLUDE

IBM default NOINCLUDE

The INCLUDE option requests the syntax checking stage of the compiler to handle the inclusion of PL/I secondary input data sets for programs that use the %INCLUDE statement. This method is faster than using the PL/I preprocessor for programs that contain %INCLUDE statements, but no other preprocessor statements. The INCLUDE option should not be used if the MACRO option also applies.

INSOURCE|NOINSOURCE

IBM default INSOURCE

The INSOURCE option specifies that the compiler is to include a listing of the source program (including preprocessor statements) in the compiler listing. This option is applicable only when the preprocessor is used; therefore the MACRO option must also apply.

INTERRUPT|NOINTERRUPT

IBM default NOINTERRUPT

This option determines the effect of attention interrupts when the compiled PL/I program is being executed.

If INTERRUPT was in effect during compilation, an established ATTENTION on-unit will be executed when one attention interrupt is caused during execution of the compiled program. If there is no such on-unit, processing will continue. Two attention interrupts will end the execution of the program and cause control to return to TSO.

If NOINTERRUPT was in effect during compilation, one attention interrupt entered during execution of the compiled program will end the execution of the program and cause control to return to TSO.

It should be noted that if any procedure within a load module was compiled with the INTERRUPT option, an attention interrupt at any time will lead to the ATTENTION condition being raised if polling is carried out, and execution continuing with no apparent effect if polling is not carried out. Polling is carried out during the execution of stream I/O for all modules, and, additionally, at branching points for modules compiled with the INTERRUPT option. Because the ATTENTION condition is raised when polling is done, an attention interrupt in a program partly compiled with the INTERRUPT option can lead to unexpected results.

A fuller discussion of the use of attention interrupts is given in Chapter 4. On a 2741 one attention interrupt is caused by pressing the ATTN key once and two by pressing it twice in quick succession. For other terminals see Appendix B.

LINECOUNT(n)

IBM default LINECOUNT(55)

The LINECOUNT option specifies the number of lines to be included in each page of the compiler listing, including heading lines and blank lines. The format of the LINECOUNT option is:

```
LINECOUNT(n)
```

where "n" is the number of lines. It must be in the range 1 through 32767, but only headings are generated if you specify less than 7.

LIST[(m[,n])]|NOLIST

IBM default NOLIST

The LIST option specifies that the compiler is to include a listing of the object module (in a form similar to IBM System/360 assembler language instructions) in the compiler listing. When used in

conjunction with MAP it increases the information generated by MAP (see MAP later in this section). "m" indicates the first statement whose assembler-language equivalent is to be listed; "n" the last. If "n" is omitted, statement "m" only is listed. If neither "m" nor "n" is specified, a listing of the whole program is produced. "m" and "n" must be line numbers if the NUMBER option applies, or statement numbers if the STMT option applies. To direct the listing of particular statements to the terminal, an option of the form LIST(m[,n]) must be specified, together with either the LIST option, with no statement numbers, within the TERMINAL option, or the PRINT(*) operand in the PLI command.

LMESSAGE|SMESSAGE

IBM default SMESSAGE

The LMESSAGE and SMESSAGE produce messages in a long form (specify LMESSAGE) or in a short form (specify SMESSAGE). Short messages can have advantages in a TSO environment due to the slow printing speed of a typewriter type terminal.

MACRO|NOMACRO

IBM default NOMACRO

The MACRO option specifies that the source program is to be processed by the preprocessor.

MAP|NOMAP

IBM default NOMAP

The MAP option specifies that the compiler is to produce tables showing the organization of the storage for the object module. These tables show how variables are mapped in the static internal control section and in DSAs, thus enabling STATIC INTERNAL and AUTOMATIC variables to be found in PLIDUMP. If LIST is also specified the MAP option produces tables showing constants, control blocks and INITIAL variable values.

LIST generates a listing of the compiled code in pseudo assembler language format. If you want a complete MAP but not a complete LIST, you can specify a single statement number as an argument for LIST to minimize the size of the LIST. For example:

```
*PROCESS MAP LIST(1);
```

MARGINI('c')|NOMARGINI

IBM default NOMARGINI

The MARGINI option specifies that the compiler is to include a specified character in the column preceding the left hand margin, and the column following the right hand margin of the listings resulting from the INSOURCE and SOURCE options. Any text in the source input which precedes the left hand margin will be shifted left one column, and any text that follows the right hand margin will be shifted right one column. Thus text outside the source margins can be easily detected.

The MARGINI option has the format:

```
MARGINI('c')
```

where "c" is the character to be printed as the margin indicator.

MARGINS(m,n,c)

IBM default: V or U-format records (10,100,0) F-format records (2,72,0)

The MARGINS option specifies which part of each compiler input record contains PL/I statements, and the position of the ANS control character that formats listings. The MARGINS option is used to override the default margin positions that are set up during compiler installation by the FMARGINS and VMARGINS options.

The FMARGINS option applies to F-format records and the VMARGINS option applies to V-format or U-format records. Only one of these defaults is overridden by the MARGINS option. If the first input record to the compiler is F-format, the FMARGINS defaults are overridden by the MARGINS option. If the first record is a V-format or U-format record, the VMARGINS defaults are overridden by the MARGINS option. Default values are assumed if a second type of record is encountered by the compiler.

The format of the MARGINS options is:

```
MARGINS(m,n,c)
```

where:

m is the column number of the leftmost column that will be scanned by the compiler. m must not exceed 100.

n is the column number of the rightmost column that will be scanned by the compiler. n must not be less then m, nor greater than 100.

c is the column of the American National Standard (ANS) printer control character. It must not exceed 100 and it must be outside the values specified for m and n. A value of 0 for c indicates that no ANS control character is present. The control character applies only to listings on a line printer; it is ignored in conversational-mode listings at the terminal. Only the following control characters can be used:

(blank)	Skip one line before printing.
0	Skip two lines before printing.
-	Skip three lines before printing.
+	Skip no lines before printing.
1	Start new page.

Any other character is taken to be blank. If the value c is greater than the maximum length of a source statement record, the compiler will not be able to recognize it; consequently, the listing will not have the required format.

MDECK[(dsname)]|NOMDECK

IBM default NOMDECK

The MDECK option specifies that the preprocessor is to produce a copy of its output on the file named SYSPUNCH. The last four bytes of the 84 byte output records are not copied, thus this option allows you to retain the output from the preprocessor as a deck of 80-column punched cards.

The data set onto which the output is written will have the name specified in "dsname." This can be a fully qualified name (enclosed in single quotation marks) or a simple name (to which the user identification and the descriptive qualifier DECK will be added). If "dsname" is not specified, the user-supplied name will be taken from the first operand of the PLI command, and the user-identification and descriptive qualifier DECK will be added. If "dsname" is not specified and the first operand of PLI specifies a member of a partitioned data set, the member name will be ignored - the generated data set name will be based on the name of the partitioned data set.

NAME

IBM default: NAME does not apply unless specified.

The NAME option specifies that the compiler is to place a linkage-editor NAME statement as the last record of the object module. When processed by the linkage editor, this NAME statement indicates that primary input is complete and causes the specified name to be assigned to the load module created from the preceding input (since any previous NAME statement).

The NAME option is required if you want the linkage editor to create more than one load module from the object modules produced by batched compilation.

If you do not use this option, the linkage editor will use the member name specified in the DD statement defining the load module data set. You can also use the NAME option to cause the linkage editor to substitute a new load module for an existing load module with the same name in the library.

The format of the NAME option is:

```
NAME('name')
```

where "name" has from one through eight characters, and begins with an alphabetic character.

NEST|NONEST

IBM default NONEST

The NEST option specifies that the listing resulting from the SOURCE option will indicate, for each statement, the begin-block level and the do-group level.

NUMBER|NONUMBER

IBM default NUMBER

The NUMBER option specifies that the numbers specified in the sequence fields in the source input records are to be used to derive the statement numbers in the listings resulting from the AGGREGATE, ATTRIBUTES, LIST, OFFSET, SOURCE, and XREF options.

The position of the sequence field can be specified in the SEQUENCE option. Alternatively, the following default positions are assumed:

- First 8 columns for V-format or U-format source input records. In this case, 8 is added to the source margins and control character position if the MARGINS option is not explicitly specified.
- Last 8 columns for F-format source input records.

These defaults are the positions used for line-numbers generated by TSO; thus it is not necessary to specify the SEQUENCE option, or change the MARGINS defaults, when using line-numbers generated by TSO.

Note that the preprocessor output has fixed-length records irrespective of the original primary input. Any sequence numbers in the primary input are repositioned in columns 73-80.

The line-number is calculated from the five right-hand characters of the sequence number (or the number of characters specified, if less than five). These characters are converted to decimal digits if necessary. Each time a line-number is found which is not greater than the preceding one, 100000 is added to this and all following line-numbers.

If there is more than one statement on a line, a suffix is used to identify the actual statement in the messages. For example, the second statement beginning on the line numbered 40 will be identified by the number 40.2. The maximum value for this suffix is 31. Thus the thirty-first and subsequent statements on a line have the same number.

If NONUMBER applies, STMT is implied. NUMBER is implied by NOSTMT or GONUMBER.

OBJECT[(dsname)]|NOOBJECT

IBM default OBJECT

The OBJECT option specifies that the compiler is to store the object module that it creates in the data set associated with the ddname SYSLIN. The data set onto which the output is written will have the

name specified in "dsname." This can be a fully qualified name (enclosed in single quotation marks) or a simple name (to which the user identification and the descriptive qualifier OBJ will be added). If "dsname" is not specified, the user-supplied name will be taken from the first operand of the PLI command, and the user-identification and descriptive qualifier OBJ will be added. If "dsname" is not specified and the first operand of PLI specifies a member of a partitioned data set, the member name will be ignored - the generated dataset name will be based on the name of the partitioned data set.

OFFSET

The OFFSET option specifies that the compiler is to include in the compiler listing a table of statement or line numbers for each procedure with their offset addresses relative to the primary entry point of the procedure. This information is of use in identifying the statement being executed when an error occurs and neither the GOSTMT nor GONUMBER option applies.

If GOSTMT applies, statement numbers, as well as offset addresses, will be included in execution-time messages. If GONUMBER applies, line numbers, as well as offset addresses, will be included in execution-time messages.

OPTIMIZE(TIME|0|2)|NOOPTIMIZE

The OPTIMIZE option specifies the type of optimization required:

NOOPTIMIZE	specifies maximum compilation speed, but inhibits optimization for faster execution and reduced main-storage requirements.
OPTIMIZE(TIME)	specifies that the compiler is to optimize the machine instructions generated for minimum execution time. A secondary effect of this type of optimization can be a reduction in the amount of main storage required for the object module. The use of OPTIMIZE(TIME) could result in a substantial increase in compile time over NOOPTIMIZE.
OPTIMIZE(0)	is the equivalent of NOOPTIMIZE.
OPTIMIZE(2)	is the equivalent of OPTIMIZE(TIME).

The language reference manual for this compiler includes a full discussion of optimization.

OPTIONS|NOOPTIONS

IBM default NOOPTIONS

The OPTIONS option specifies that the compiler is to include, in the compiler listing, a list showing the compiler options to be used during this compilation. This list includes all options applied by default.

If the PRINT(*) operand of the PLI command applies, the list of options will be printed at the terminal. This will show the negative forms of

the OPTION option and all other options that cause listings to be produced, even where the positive forms in fact apply. The positive forms will be shown within the TERMINAL option. This is because the PRINT(*) operand is implemented by generating a TERMINAL option containing a list of options corresponding to those listings that are to be printed at the terminal.

SEQUENCE(m,n)|NOSEQUENCE

IBM default: F-format records (73,80)
V- and U-format record (1,8)

The SEQUENCE option is concerned with the position of the sequence numbers in compiler input records. It specifies the positions from which the compiler will extract sequence numbers It is used to override the defaults that are set up at compiler installation time by the FSEQUENCE and VSEQUENCE options. The values specified in the SEQUENCE option override the default values for FSEQUENCE, if the first record read is an F-format record, and VSEQUENCE if it is a V-format or U-format record. If the input to the compiler contains both F-format and V-format or U-format records, the values set up during compiler installation will apply to the second type of record. The number found in the sequence field is included in the source listings produced by the FORMAT, INSOURCE, and SOURCE options. If the NUMBER option applies, statement numbers are derived from the sequence numbers. No attempt is made to sort the input records into the sequence implied by the numbers. The SEQUENCE option has the format:

SEQUENCE(m,n)

where: m specifies the column number of the leftmost digit of the statement number.

n specifies the column number of the rightmost digit of the statement number.

SIZE(yyyyyyyy|yyyyyK|MAX)

IBM default SIZE(MAX)

The SIZE option can be used to limit the amount of main storage used by the compiler. This is of value, for example, when dynamically invoking the compiler, to ensure that space is left for other purposes.

The SIZE option can be expressed in three forms:

SIZE(yyyyyyyy) specifies that yyyyyyyy bytes of main storage are to be requested. Leading zeros need not be specified.

SIZE(yyyyyK) specifies that yyyyyK bytes of main storage are to be requested (1K=1024). Leading zeros need not be specified.

SIZE(MAX) specifies that the compiler is to obtain as much main storage as it can.

The IBM default, and the most usual value to be used, is SIZE(MAX), which permits the compiler to use as much main storage in the region as it can.

When a limit is specified, the amount of main storage used by the compiler depends on how the operating system has been generated, and the method used for storage allocation. The compiler assumes that buffers, data management routines, and processing phases take up a fixed amount of main storage, but this amount can vary undetected by the compiler. Under MVT, a region of 56K bytes or more is required.

After the compiler has loaded its initial phases and opened all files, it attempts to allocate space for working storage. If SIZE(MAX) is specified it obtains all space remaining in the region (after allowance for subsequent data-management storage areas). If a limit is specified then this amount of storage is requested. If the amount available is less than specified, but is more than the minimum workspace required, compilation proceeds. If insufficient storage is available, compilation is terminated. This latter situation should arise only if the region is too small, that is, less than 56K, or if too much space for buffers has been requested. The value cannot exceed the main storage available for the job step and cannot be changed after processing has begun. This means that, in a batched compilation, the value established when the compiler is invoked cannot be changed for later programs in the batch. Thus it is ignored if specified in a *PROCESS statement after the first program.

An additional 10K to 30K bytes must be allowed for TSO. The actual size required for TSO depends on which routines are placed in the link-pack area (a common main storage pool available to all regions). Also, if you run the compiler in TSO edit mode, about 30K bytes are required for the EDIT routines.

SMESSAGE

See LMESSAGE option.

SOURCE|NOSOURCE

IBM default NOSOURCE

The SOURCE option specifies that the compiler is to include a source program listing in the compiler listing. The source program listed is either the original source input or, if the MACRO option applies, the output from the preprocessor.

STMT|NOSTMT

IBM default NOSTMT

The STMT option specifies that statements in the source program are to be counted, and that the resulting statement numbers are to be used to identify statements in the listings resulting from the AGGREGATE, ATTRIBUTES, LIST, OFFSET, SOURCE, and XREF options. If NOSTMT applies, NUMBER is implied. STMT is implied by NONUMBER or GOSTMT.

STORAGE|NOSTORAGE

IBM default NOSTORAGE

The STORAGE option specifies that the compiler is to include in the compiler listing a table giving the main storage requirements for the object module.

SYNTAX|NOSYNTAX[(W|E|S)]

IBM default NOSYNTAX(S)

The SYNTAX option specifies that the compiler is to continue into syntax checking after initialization (or after preprocessing if the MACRO option applies) unless an unrecoverable error is detected. The NOSYNTAX option without an argument causes processing to stop unconditionally after initialization (or preprocessing). With an argument, continuation depends on the severity of errors detected so far, as follows:

NOSYNTAX(W)	No syntax checking if a warning, error, severe error, or unrecoverable error is detected.
NOSYNTAX(E)	No syntax checking if an error, severe error, or unrecoverable error is detected.
NOSYNTAX(S)	No syntax checking if a severe error or unrecoverable error is detected.

If the SOURCE option applies, the compiler will generate a source listing even if syntax checking is not performed.

TERMINAL[(option-list)]|NOTERMINAL

IBM default TERMINAL

The TERMINAL option specifies that some, or all of the compiler listing produced during compilation is to be copied at the terminal. If TERMINAL is specified without an argument, any diagnostic and informatory messages are printed at the terminal. You can add an argument, which takes the form of an option list, to specify other parts of the compiler listing that are to be printed at the terminal.

The listing at the terminal is independent of that written on SYSPRINT. However, if SYSPRINT is associated with the terminal, only one copy of each option requested will be printed even if it is requested in the TERMINAL option and also as a first-level option.

The following option keywords, their negative forms, or their abbreviated forms, can be specified in the option list for the TERMINAL option:

AGGREGATE, ATTRIBUTES, ESD, INSOURCE, LIST, MAP, OFFSET, OPTIONS, SOURCE, STORAGE, and XREF.

If the option does not apply to the compiler listing, specifying it in the TERMINAL option has no effect.

The other options that relate to the listing (that is, FLAG, LMESSAGE, MARGINI, NEST, NUMBER, STMT and the SHORT and FULL suboptions of ATTRIBUTES and XREF) will be the same as for the listing on the standard output file.

XREF[(SHORT|FULL)]|NOXREF

IBM default NOXREF
IBM default suboption FULL

The XREF option specifies that the compiler is to include in the compiler listing a list of all identifiers used in the PL/I program, together with the numbers of the statements in which they are declared or referenced. Refer to the programmer's guide listed in the Preface for a description of the format and content of the cross-reference table.

If the suboption SHORT is specified, unreferenced identifiers are not listed, making the listing more manageable.

If both XREF and ATTRIBUTES are specified, the two listings are combined. If there is a conflict between SHORT and FULL, the usage is determined by the last option specified. For example, ATTRIBUTES(SHORT) XREF(FULL) results in FULL applying to the combined listing.

Appendix A: Command Syntax

SYNTAX NOTATION

The syntax notation used to illustrate the commands and subcommands in this manual follows that in used and explained in the manual OS/VS2 TSO Command Language Reference. Briefly, the conventions are as follows:

Items in brackets [ ] are optional

Items stacked between braces { } are alternatives; choose only one.

Items separated by a logical OR sign | are alternatives; choose only one. (This convention is not used in the Command Language Reference manual.)

Items in capital letters are keywords. The command or subcommand name must be spelled as shown. Operands of commands can be shortened by deleting any number of characters from the right, provided the result is unambiguous.

Items in lowercase letters must be replaced by appropriate names or values.

The special characters ' ( ) * must be included where shown

COMMAND SYNTAX

A command or subcommand consists of a keyword, followed, in the general case, by several operands. The rules for entering a command or subcommand are fully described in the manual OS/VS2 TSO Command Language Reference. The following is a summary of the general syntax rules.

Separate the command or subcommand name from the first operand by one or more blanks.

Separate operands by a blank or a comma. You can insert any number of extra blanks around the mandatory blank or the comma.

Positional operands, shown in lowercase letters in the syntax notation, must follow the command or subcommand keyword in the prescribed order. Keyword operands, shown in capital letters, must follow the positional operands, but may be in any order; they must be typed as shown.

End a subcommand with a semicolon if it is followed by any other input on the same line.

Appendix B: Using a Terminal Other Than a 2741

If you are using a terminal other than a 2741, you should regard the appropriate terminal manual as the source for information on how to use it. If it is an IBM terminal, it will be described in the manual OS/MVT and OS/VS2 TSO: Terminals, Order No. GC28-6762.

In principle, this appendix describes only those items that are of particular relevance to PL/I programming. These are, causing an attention interrupt from the terminal, and various formatting anomalies that occur when using display-screen type terminals. However, when this manual was originally written it included general information on how to use the 2260 and 2265 display screens and this information is retained for the benefit of those who are used to using it as an information source.

CAUSING AN ATTENTION INTERRUPT

An attention interrupt in an optimizer compiled procedure causes the program to enter its attention exit. A description of how to cause an attention interrupt is given below for the 3277, 2260, and 2265 display screens, for other terminals the appropriate terminal and system documentation must be read.

ATTENTION ON 3277, 2260, AND 2265 DISPLAY SCREENS

Before an attention interrupt can be caused on a 3277, 2260, or 2265 display screen the system must have issued a READ to the terminal. During execution of the program, READs are only issued when the screen is full, when the optimizer expects input, or at intervals specified in the TERMINAL command. On a 3277, this is when the INPUT INHIBITED light is off.

To make full use of the compiler, it is necessary to use the TERMINAL command to set points where TSO will stop execution and give you the opportunity of entering an attention interrupt. On the 2260 and 2265 it is also necessary to set an input string that will simulate the attention condition and though not strictly necessary, this also has some advantages on a 3277. Like setting the time interval, it is done using the TERMINAL command.

USING THE TERMINAL COMMAND

A simplified form of the TERMINAL command showing only the relevant options is:

```
TERMINAL SECONDS(n) LINES(n) INPUT(attention_string)
```

SECONDS(n) specifies the amount of time in seconds that will elapse without terminal activity taking place before an intervention request is issued by TSO enabling you to enter an attention interrupt. n must be a multiple of 10 seconds, if it is not, it is rounded up to the nearest multiple of ten. The maximum value is 2550.

LINES(n) specifies the number of lines that will be displayed at the terminal before an intervention request is issued. n can be from 1 to 255, however you should remember that an intervention request is always issued when the screen is full, so there is no point in making n larger than the screen size.

INPUT(attention_string)
specifies the string that will be used for causing an attention interrupt. This is essential on the 2260 and 2265 and has some advantages on the 3277. Attention_string is one to four non-blank characters.

For example:

```
TERMINAL SECONDS(30) INPUT(ATTN)
```

means that TSO will issue an intervention request if 30 seconds pass without input from, or output to the terminal, and that the string entered in response to the intervention request to simulate attention will be ATTN. (30 seconds is a useful time interval to try as a start giving adequate control without excessive interruption of processing.) Note that a TERMINAL command may form part of your LOGON procedure. You should check with your systems programmer.

RESPONDING TO AN INTERVENTION REQUEST

When an intervention request has been issued, TSO puts out the string "***" followed by a cursor mark, and processing is halted. You may continue normal processing by pressing the enter key (on the 2260 you must hold down the SHIFT key at the same time). Alternatively you may issue an attention interrupt in one of the following ways:

1. Entering the string specified in the TERMINAL command thus:
   ***ATTN

2. Entering the string followed by a digit from 1 to 9. This controls the number of attention interrupts that will be caused. 1 will pass control to an established attention on-unit or be ignored if there is not one. 2 will return control to the next higher system, normally TSO, thus:
   ***ATTN2

3. Enter the digit without the string. The digit has the effect described above and can be used on the 3277 regardless of whether an attention string has been specified thus:
   ***1

4. 3277 only, press the PA1 key thus:
   *** (you press the PA1 key)

Attention when the PL/I program expects input
When input is expected from the terminal, (that is, when it has issued a prompt ending a:), an attention interrupt can be specified in any of the ways above except number 3. On the 3277, this means that there is an advantage in specifying an attention simulation string because the number of interrupts can then be specified in this situation by using method 2. This enables you to exit easily from your program into TSO.

FORMATTING ON A 3277 DISPLAY SCREEN

Certain anomalies arise when formatting output on a 3277 display screen. When stream output is transmitted to the terminal, it is preceded by an attribute character which contains information such as the intensity required for the item. This attribute character appears on the screen as a blank character.

Thus, for example, the statement PUT(EDIT('A','B')(A); formats thus:

```
AB
```

whereas the two statements PUT EDIT('A')(A); PUT EDIT('B')(A); format thus:

```
A B
```

This can result in incorrect alignment of tables, and other apparent anomalies.

Further troubles may occur when the size of a transmitted item approaches screen width, because the transmitted item and the attribute character may take up more than one line on the screen.

Appendix C: %INCLUDE Data

SECONDARY INPUT

In conversational mode, as in batch mode, you can incorporate PL/I source code into your program by means of the %INCLUDE statement. The statement names members of partitioned data sets that hold the code to be included. You can create these secondary input data sets either under TSO or in batch mode.

%INCLUDE is a preprocessor statement; to invoke the preprocessor, the MACRO compiler option must apply. However, if there are no other preprocessor statements in your program, you need not invoke the preprocessor: the syntax checking stage of the compiler will include the secondary input, if you specify the option INCLUDE.

The %INCLUDE statement can specify simply the name of the data set member that holds the text to be included. For instance:

```
%INCLUDE RECDCL;
```

It can also specify a ddname that is to be associated with the member. For instance:

```
%INCLUDE STDCL(F726);
```

STDCL is the ddname, and F726 is the member name. A single %INCLUDE statement can specify several data set members, and can contain both forms of specification. For instance:

```
%INCLUDE SUBA(READ5),SUBC(REPORT1),DATEFUNC;
```

ALLOCATING DATA SETS

All data sets containing secondary input must be allocated before the compiler is invoked.

If a data set member name is specified in a %INCLUDE statement without a ddname, then the data set can be allocated by specifying the data set name in the LIB operand of the PLI command. (This operand is the equivalent of the batch-mode SYSLIB DD statement.) The necessary allocation will be made by the PL/I prompter.

If a ddname has been specified in the %INCLUDE statement, the corresponding data set must be allocated by means of either an ALLOCATE command or the logon procedure.

Suppose, for example, that the data set members specified in the %INCLUDE statements in the preceding section are held on data sets as follows (the ddname used in the %INCLUDE statement is also shown):

Member:	Data set name:	ddname:
RECDCL	LDSRCE	none
F726	WPSRCE	STDCL
READ5	JESRCE	SUBA
REPORT1	GHSRCE	SUBC
DATEFUNC	DRSRCE	none

Then the necessary data sets could be allocated by the following commands.

```
allocate file(stdcl) dataset(wpsrce)
allocate file(suba) dataset(jesrce)
allocate file(subc) dataset(ghsrce)
pli mnthcost lib(ldsrce,drsrce) include
```

INPUT RECORD FORMATS

The compiler will accept both F-format and V-format records, and the primary and secondary input data sets can have different formats.

The compiler determines the positions, within each record, of the PL/I source code and the sequence numbers from the following options.

Option	Specifying	Standard default
FMARGINS FSEQUENCE	Positions of source text and sequence numbers for F-format records	FMARGINS(2,72) FSEQUENCE(73,80)
VMARGINS VSEQUENCE	Positions of source text and sequence numbers for V-format records	VMARGINS(10,100) VSEQUENCE(1,8)
MARGINS SEQUENCE	Overriding values for above options	- -

The values of FMARGINS, FSEQUENCE, VMARGINS, and VSEQUENCE can be set only at system generation time. If no values are set at this time, the standard default values apply. MARGINS and SEQUENCE can be specified when the compiler is invoked. When specified, they override either FMARGINS and FSEQUENCE or VMARGINS and VSEQUENCE, depending on whether the first input data set read by the syntax-checking stage of the compiler is F-format or V-format. The overriding values will also apply if records of the same format are read as secondary input. If records of the other format are read as secondary input, the system generation values for that format apply.

SOURCE LISTINGS AND STATEMENT NUMBERS

If the MACRO option applies, the source listing will show the included text in place of the %INCLUDE statements in the primary input data set.

If the MACRO option does not apply but the INCLUDE option does, the source listing will contain both the %INCLUDE statements and the included text. Each piece of included text will immediately follow the corresponding %INCLUDE statement. The end of the included text will be marked by a line of asterisks; with F-format primary input, the line containing the %INCLUDE statement will also have asterisks between the end of the statement and the right hand margin.

If the STMT compiler option applies, the statement numbers are derived from a count of the number of statements in the program after secondary input has been included.

If the NUMBER option applies, the statement numbers are derived from the sequence numbers of the primary and secondary input records. Normally the compiler uses the last five digits as statement numbers. If, however, this would give numbers such that each statement does not have a higher number than the one before it, 100000 is added to all statement numbers starting from the one which would otherwise be equal to or less than its predecessor.

For instance, if a V-format primary input data set had the following lines:

```
00001000 A:PROC;
00002000 %INCLUDE B;
00003000 END;
```

and member B contained:

```
00001000 C=D;
00002000 E=F;
00003000 G=H;
```

then the source listing would be as follows.

```
SOURCE LISTING
    NUMB
    1000 00001000 A:PROC;
         00002000 %INCLUDE B;
  101000 00001000 C=D;
  102000 00002000 E=F;
  103000 00003000 G=H;
                  ***********
  203000 00003000 END;
```

The additional 100000 has been introduced into the statement numbers at two points: once beginning at the first statement of the included text (the statement C=D;) and again beginning with the first statement after the included text (the END statement.)

OS PL/I Optimizing Compiler:
TSO User's Guide
SC33-0029-3

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL _____

Previous TNL _____

Previous TNL _____

Fold on two lines, tape, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

OS PL/I Optimizing Compiler: TSO User's Guide (File No.S360/S370-29) Printed in U.S.A. SC33-0029-3

SC33-0029-3

OS PL/I Optimizing Compiler: TSO User's Guide (File No.S360/S370-29) Printed in U.S.A. SC33-0029-3

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601