PL/I

TECHNICAL REPORT

ULD VERSION III

TR 25.096
30 June 1969

CONCRETE SYNTAX OF PL/I

G. URSCHLER

IBM LABORATORY VIENNA

N O T E

This document is not an official PL/I Language Specification. For information concerning the official interpretation the reader is referred to the PL/I Language Specifications, Form No. Y33-6003-1.

IBM LABORATORY VIENNA,
Austria

CONCRETE SYNTAX OF PL/I

by

G. URSCHLER

ABSTRACT

This report supplements the semantical definition of PL/I given in "Abstract Syntax and Interpretation of PL/I" and the specification of abstract syntax given in "Translation of PL/I into Abstract Syntax" (IBM Laboratory Vienna, TR 25.098 and TR 25.097) by a syntactical definition. The syntactical form of concrete PL/I program text is defined by means of an extended Backus notation, which is described by a meta syntax.

Locator Terms for IBM Subject Index

PL/I
Backus Notation
Formal Definition
Syntax, concrete
21 PROGRAMMING

TR 25.096

30 June 1969

PREFACE

This document is an updated version of:

/1/ ALBER, K., OLIVA, P., URSCHLER, G.: Concrete Syntax of PL/I.
IBM Laboratory Vienna, Techn. Report TR 25.084, 28 June 1968.

It is part of a series of documents (ULD Version III) presenting the formal definition of syntax and semantics of PL/I:

/2/ FLECK, M.: Formal Definition of the PL/I Compile Time Facilities (ULD Version III).
IBM Laboratory Vienna, Techn. Report TR 25.095, 30 June 1969.

/3/ URSCHLER, G.: Concrete Syntax of PL/I (ULD Version III).
IBM Laboratory Vienna, Techn. Report TR 25.096, 30 June 1969.

/4/ URSCHLER, G.: Translation of PL/I into Abstract Text (ULD Version III).
IBM Laboratory Vienna, Techn. Report TR 25.097, 30 June 1969.

/5/ WALK, K., ALBER, K., FLECK, M., GOLDMANN, H., LAUER, P., MOSER, E., OLIVA, P., STIGLEITNER, H., ZEISEL, G.: Abstract Syntax and Interpretation of PL/I (ULD Version III).
IBM Laboratory Vienna, Techn. Report TR 25.098, 30 April 1969

/6/ ALBER, K., GOLDMANN, H., LAUER, P., LUCAS, P., OLIVA, P., STIGLEITNER, H., WALK, K., ZEISEL, G.: Informal Introduction to the Abstract Syntax and Interpretation of PL/I (ULD Version III).
IBM Laboratory Vienna, Techn. Report TR 25.099, 30 June 1969.

The method and notation used in these documents are essentially taken over from the first version of a formal definition of PL/I issued by the Vienna Laboratory:

/7/ PL/I Definition Group of the Vienna Laboratory: Formal Definition of PL/I.
IBM Laboratory Vienna, Techn. Report TR 25.071, 30 December 1966

/8/ ALBER, K.: Syntactical Description of PL/I Text and its Translation into Abstract Normal Form.
IBM Laboratory Vienna, Techn. Report TR 25.074, 14 April 1967.

An outline of the method is given in:

/9/ LUCAS, P., LAUER, P., STIGLEITNER, H.: Method and Notation for the Formal Definition of Programming Languages.
IBM Laboratory Vienna, Techn. Report TR 25.087, 28 June 1968.

This document also contains the appropriate references to the relevant literature. The basic ideas and their application to PL/I have been made available through several workshops on the formal definition of PL/I, and presentations and publications inside and outside IBM. The method is demonstrated by application to an appropriately tailored subset of PL/I in:

/10/ LUCAS, P., WALK, K.: On the Formal Description of PL/I.
To be published in Annual Review in Automatic Programming - Vol.6.
Pergamon Press, New York 1969.

The language defined in the present version is PL/I as specified in the PL/I Language Specifications, Form No. Y33-6003-1, with the addition of attention handling, input stream and string scanning, and file variables.

The present document will be made subject to validation by the PL/I Language Department, Hursley.

PRODUCTION

This document was prepared by means of automated text-processing systems. TEXT 360 was used for processing the prose parts. The formatting, indexing, cross-referencing, and updating of formula texts was handled by means of FORMULA 360.

FORMULA 360 is a syntax-controlled formula processing system which was developed in the Vienna Laboratory especially to facilitate the production and maintenance of PL/I Formal Definition documents. The achievements of K.F. KOCH in the overall design and implementation of FORMULA 360 are acknowledged in particular. Essential components of the system are due to G. URSCHLER (syntactical decomposition of formulas) and E. MOSER (formula input checker). H. Hoja and G. Zeisel contributed to the clarification and formulation of the required formatting processes.

Coordination: F. Schwarzenberger, M. Stadler

Technical control: K.F. Koch, E. Moser, M. Stadler

Data transcription: Miss W. Schatzl, Mrs. H. Deim, and sub-contractors

System support: H. Chladek, G. Lehmayer

CONTENTS

1. INTRODUCTION

This document contains a formal description of the concrete syntax of PL/I.

The syntax is described by giving a generation process for concrete program texts (section 2.3), which is based on a context free production system. To facilitate the description of inserting spaces, the production system is divided into two parts, a higher and a lower level syntax (sections 3.1 and 3.2, respectively).

The production rules are written in extended Backus notation. The syntactical form and the meaning of this notation are explained in chapter 2.

It is the intent of this paper to present a syntax which minimizes the syntactic complexity of PL/I programs. As a consequence, the syntax is rather permissive, in the sense that it allows the production of a great number of non-sensical programs. This design aim has been adopted for the following reasons:

(1) The syntactic description presented should be easily readable and understandable.

(2) The syntactic description should be natural in the sense that the syntactic components are the meaningful components of a program. Adding more syntactic restrictions can easily spoil the clean structure of programs.

(3) Additional syntactic restrictions, in particular context-dependent restrictions, can more easily be handled in the Translator (/4/) or even in the Interpreter (/5/).

(4) The syntax in its present form is suitable for a test on non-ambiguity.

2. SYNTAX NOTATION

2.1 SEMANTICS OF THE EXTENDED BACKUS NOTATION

The Backus notation as used in the Algol 60 Report is slightly modified in this paper. The printed brackets are replaced by spaces, so that a production rule gets the general form:

V ::= S_1 | S_2 | ... | S_n

(V is to be replaced by one of the alternatives S_1 or S_2 or ... or S_n)

Note: In this chapter with V variables, with S_n ($n \in \{1,2,3,...\}$) arbitrary strings and with T_n strings different from the null-string are denoted. Each of these strings may consist of a certain number of not nearer specified syntactical units, denoted by U_n.

As a further convenience we introduce the extended Backus notation, which has been developed by both the Hursley and the Vienna Laboratories.

This notation uses beyond '::=' and '|' the metalinguistic signs '{', '}', '[', ']', '.' and '...' with the following meaning:

(1) A production like

goto-statement ::= GOTO reference ; | GO TO reference ;

may be shortened to

goto-statement ::= { GOTO | GO TO } reference ;

In general

V ::= S_1 T_1 S_2 | S_1 T_2 S_2 | ... | S_1 T_n S_2

may be replaced by

V ::= S_1 { T_1 | T_2 | ... | T_n } S_2

and vice versa. (Notice that this rule remains valid also for the case n=1)

(2) A production like

return-statement ::= RETURN ; | RETURN (expression) ;

may be shortened to

return-statement ::= RETURN [(expression)] ;

In general

V ::= S_1 S_2 | S_1 T_1 S_2 | ... | S_1 T_n S_2

may be replaced by

V ::= S_1 [T_1 | T_2 | ... | T_n] S_2

and vice versa.

(3) A production like

integer ::= digit | integer digit

may be shortened to

integer ::= digit•••

In general

V ::= U | V U or V ::= U | U V

may be replaced by

V ::= U••• and vice versa.

Note: The production V ::= U••• is often omitted and instead of V then always U••• is written. This signifies for the inverse process of eliminating all instances of U•••, that they must be replaced by a newly introduced variable, say V, and one of the productions V ::= U | U V or V ::= U | V U is to be added to the other production rules.

(4) A production like

declarationlist ::= declaration [{ , declaration }•••]

may be shortened to

declarationlist ::= { , • declaration••• }

In general

V ::= S_1 T_1 [{ T_2 T_1 }•••] S_2

may be replaced by

V ::= S_1 { T_2 • T_1••• } S_2 and vice versa.

Note: Instead of S_1 [{ T_2 • T_1••• }] S_2 also S_1 [T_2 • T_1•••] S_2 may be written.

2.2 THE SYNTAX OF THE EXTENDED BACKUS NOTATION

The expressions "syntactical unit" and "string of syntactical units" (called "unit" and "sequence" respectively in the following) have not been specified in the last section. We now define them together with the general form of the production rules of the concrete PL/I syntax recursively by means of a meta syntax. The meta syntax itself is written in Backus form.

To uphold the Backus notation also formally we use the metalinguistic signs '::=' and '|' in the meta syntax. Therefore we are obliged to change the notation for similar syntactic or PL/I signs. We adopt the same convention as in chapter 3, that each ambiguous sign is marked on the lower syntactic level by a further underlining. This signifies, for this and only for this section, that the PL/I signs for colon, equal and or-sign get the forms '$\underline{:}$', '$\underline{=}$', '$\underline{|}$' while the or-sign of the PL/I production rules is denoted by '|'.

Meta Syntax

```
prod-rule      ::= not-var ::= definition

definition     ::= sequence | sequence | definition

sequence       ::= unit | unit sequence

unit           ::= not-var | not-const | unit••• |
                   { definition } | [ definition ] |
                   { not-const • unit••• } | [ not-const • unit••• ]

not-var        ::= sm-letter | sm-letter - not-var | sm-letter not-var

sm-letter      ::= a | b | c | d | e | f | g | h | i | j | k | l | m |
                   n | o | p | q | r | s | t | u | v | w | x | y | z

not-const      ::= PL/I-symb | PL/I-symb not-const

PL/I-symb      ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
                   N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
                   $ | @ | # | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
                   blank | _ | = | + | - | * | / | ( | ) | , | . | ; |
                   : | & | | | ¬ | > | < | ? | % | '
```

Auxiliary rules for the insertion of spaces:

Spaces (e.g., blanks, new lines, new pages) are optional immediately preceding or succeding '::=' or '|' or '{' or '}' or '[' or ']' or '•' or '•••' and between arbitrary adjacent units. A space is mandatory (to avoid ambiguities) between adjacent not-vars and between adjacent not-consts.

2.3 GENERATION OF A CONCRETE PROGRAM TEXT

2.3.1 THE NORMAL GENERATION PROCESS

First of all any implementation must provide production rules for the four implementation dependent notation variables external-option, env-option, incorporate-specification, and extralingual-character. Since PL/I has context dependent rules for the insertion of blanks and comments, which cannot be expressed by production rules of the form described in 2.1 and 2.2, the generation of a concrete PL/I program text has to be performed in four steps:

(1) Starting with the notation variable "program", replacements are to be performed according to the higher level production rules listed in 3.1. This process is to be continued as long as any higher level production rule is applicable.

It ends up with a text consisting of "PL/I words", which are listed in 3.3. In this respect, all those sequences of PL/I symbols which in the production rules are not separated by empty space are assumed to compose words (notation constants) and not to be split up into their single symbols.

So a word is one of the following:

a single PL/I symbol,

a keyword, which is a sequence of upper case letters,

one of the eight composite operators:

** || >= <= ¬> ¬= ¬< ->,

one of the eight notation variables

identifier,
integer,
isub,
real-constant,
imaginary-constant,
simple-string-constant,
sterling-constant,
picture-specification.

(2) Now "spaces" are inserted into the generated text according to the following rule:

The 24 words

= + - * / ( ) , . ; : & | ¬ < > ** || >= <= ¬> ¬= ¬< ->

are "delimiters", all other words "non-delimiters". Between two adjacent non-delimiters the notation variable "space" must be inserted, between other combinations of words or following the last word of the complete program the notation variable "space" may be inserted.

The production rules for "space" are listed in 3.2.3.

(3) Now the replacement is continued by application of the lower level production rules listed in 3.2.

(4) Finally all notation constants are split up into their single symbols.

The complete process ends up with a text consisting of the symbols of the 60 character PL/I alphabet, i.e.,

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z $ @ #
0 1 2 3 4 5 6 7 8 9 _ blank = + - * / ( ) , . ; : ' & | ¬ > < ? %

and extralingual characters.

2.3.2 AUXILIARY RULES FOR ADDITIONAL FACILITIES

PL/I contains two facilities which in the one case would lengthen unnecessarily the production rules and in the other case cannot be expressed by context independent production rules. Both facilities allow a program text to be replaced by a shorter one, without changing the semantical meaning.

2.3.2.1 Keyword abbreviations

The following abbreviations may be inserted instead of the corresponding keywords. This replacement has to be performed before step 3 of the generation process described in 2.3 is performed:

keywords:	abbreviations:
ATTENTION	ATTN
AUTOMATIC	AUTO
BCOLUMN	BCOL
BEGINVOLUME	BOV
BINARY	BIN
BUFFERED	BUF
CHARACTER	CHAR
COLUMN	COL
COMPLEX	CPLX
CONNECTED	CONN
CONTROLLED	CTL
CONVERSION	CONV
DECIMAL	DEC
DECLARE	DCL
DEFINED	DEF
ENDVOLUME	EOV
ENVIRONMENT	ENV
EXCLUSIVE	EXCL
EXTERNAL	EXT
FIXEDOVERFLOW	FOFL
INITIAL	INIT
INTERNAL	INT
IRREDUCIBLE	IRRED
NOCONVERSION	NOCONV
NOFIXEDOVERFLOW	NOFOFL
NOOVERFLOW	NOOFL
NOSTRINGRANGE	NOSTRG
NOSUBSRIPTRANGE	NOSUBRG
NOUNDERFLOW	NOUFL
NOZERODIVIDE	NOZDIV
OVERFLOW	OFL
PICTURE	PIC
POINTER	PTR
POSITION	POS
PROCEDURE	PROC
REDUCIBLE	RED
SEQUENTIAL	SEQL
STRINGRANGE	STRG
STRINGSIZE	STRZ
SUBSRIPTRANGE	SUBRG
UNALIGNED	UNAL
UNBUFFERED	UNBUF
UNDEFINEDFILE	UNDF
UNDERFLOW	UFL
VARYING	VAR
ZERODIVIDE	ZDIV

2.3.2.2 Multiple closure of blocks and groups

Assume, that all four steps of the generation process described in 2.3.1 including the insertion of abbreviated keywords have been terminated.

Then a part of this program text is called a compound if it could have been generated by means of the following production rule:

```
compound ::= procedure | [ prefixlist ] [ labellist ]
             { begin-block | group }
```

Before the rightmost semicolon of a compound, i.e., between 'END' and ';', one identifier of the labellist of the compound (i.e., the labellist before its leftmost 'PROCEDURE' or 'BEGIN' or 'DO' keyword) may be inserted.

Provided that a compound actually has such an end, e.g., 'END IDENTIFIER ;' it is allowed to omit an immediately preceding 'END ;' if the inserted identifier does not occur in the labellist of the compound which is closed by the end-clause to be omitted.

2.3.3 PROGRAMS IN THE 48 CHARACTER SET

It is possible to write PL/I programs in the following 48 character set:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
$ 0 1 2 3 4 5 6 7 8 9 blank = + - * / ( ) . , '
```

If the program shall be written in this character set, in addition to the processes described in 2.3.1 and 2.3.2 the following rules have to be obeyed:

(1) From the production rules for "letter", "alphameric-character", "string-character" and "comment-symbol" the following 12 symbols have to be deleted:

```
@ # _ ; : & | ¬ > < ? %
```

(2) The following 13 PL/I words have to be handled as notation variables and to be replaced by means of the (higher level) production rules:

```
; ::= ,.                 >= ::= GE
¬ ::= NOT                <= ::= LE
& ::= AND                ¬> ::= NG
| ::= OR                 ¬< ::= NL
> ::= GT                 ¬= ::= NE
< ::= LT                 || ::= CAT
                         -> ::= PT
```

For the insertion of spaces the word ',. ' is handled as a delimiter and the other 12 words resulting from these replacements as non-delimiters.

(3) The 12 sequences of letters

NOT, AND, OR, GT, LT, GE, LE, NG, NL, NE, CAT, PT

are "reserved words", i.e., no identifier must finally be replaced by any of these sequences.

(4) In the final text, each colon ':' is to be replaced:

(a) when immediately following a dot '.' by means of the production rule

```
: ::= space . .
```

(b) else by means of the production rule

```
: ::= . .
```

3. CONCRETE SYNTAX

3.1 HIGHER LEVEL PRODUCTIONS

(1) program ::=

procedure•••

(2) procedure ::=

[prefixlist] labellist PROCEDURE [parameterlist] [procedure-optionslist] ; sentencelist

(3) parameterlist ::=

([, • identifier•••])

(4) procedure-optionslist ::=

{ options-attribute | returns-attribute | ORDER | REORDER | RECURSIVE }•••

(5) sentencelist ::=

[sentence•••] end-clause

(6) end-clause ::=

[prefixlist] [labellist] END ;

(7) sentence ::=

procedure | entry | declaration-sentence | format-sentence | statement

(8) entry ::=

labellist ENTRY [parameterlist] [returns-attribute] ;

3.1.1 DECLARATIONS

(9) declaration-sentence ::=

[labellist] { declare-sentence | default-sentence }

(10) declare-sentence ::=

DECLARE declarationlist ;

(11) declarationlist ::=

{ , • declaration••• }

(12) declaration ::=

[integer] { identifier |
(declarationlist) } [dimension-attribute] [attribute•••]

(13) default-sentence ::=

default-option-1 | default-option-2

(14) default-option-1 ::=

DEFAULT ALL [attribute-spec] ;

(15) default-option-2 ::=

DEFAULT { , • default-spec••• } ;

(16) default-spec ::=

simple-default-spec | factored-default-spec

(17) simple-default-spec ::=

range-spec [attribute-spec]

(18) range-spec ::=

identifier-range-spec | DESCRIPTORS

(19) identifier-range-spec ::=

RANGE ([[, • { identifier | letter : letter }••• } | * })

(20) attribute-spec ::=

[dimension-attribute] { attribute | value-clause }••• | SYSTEM

(21) value-clause ::=

VALUE ({ , • value-spec••• })

(22) factored-default-spec ::=

({ , • default-spec••• }) [attribute-spec]

(23) value-spec ::=

precision-spec | string-attribute | area-attribute

(24) precision-spec ::=

```
arithmetic-attribute••• |
( { , • arithmetic-attribute••• } ) arithmetic-attribute
```

3.1.1.1 Attributes

(25) options-attribute ::=

```
OPTIONS ( { , • external-option••• } )
```

(26) returns-attribute ::=

```
RETURNS ( { data-attribute | entry-name-attribute | FILE }••• )
```

(27) dimension-attribute ::=

```
( { , • bound-pair••• } )
```

(28) bound-pair ::=

```
[ refer-expression : ] refer-expression | *
```

(29) refer-expression ::=

```
expression [ REFER ( unsubscripted-reference ) ]
```

(30) attribute ::=

```
data-attribute | non-data-attribute | entry-name-attribute |
file-name-attribute | scope-attribute | like-attribute
```

(31) data-attribute ::=

```
arithmetic-attribute | string-attribute | VARYING | picture-attribute |
area-attribute | label-attribute | POINTER | offset-attribute | TASK |
EVENT | storage-class-attribute | defined-attribute | based-attribute |
UNALIGNED | ALIGNED | SECONDARY | CONNECTED | VARIABLE | initial-attribute
```

(32) arithmetic-attribute ::=

```
{ REAL | COMPLEX | DECIMAL | BINARY | FLOAT |
FIXED } [ ( integer [ , signed-integer ] ) ]
```

(33) signed-integer ::=

```
[ + | - ] integer
```

(34) string-attribute ::=

```
{ BIT | CHARACTER } [ ( { refer-expression | * } ) ]
```

(35) picture-attribute ::=

```
PICTURE [ picture-specification ]
```

```
(36)   area-attribute ::=

          AREA [ ( { refer-expression | * } ) ]

(37)   label-attribute ::=

          LABEL [ ( { , • identifier••• } ) ]

(38)   offset-attribute ::=

          OFFSET [ ( reference ) ]

(39)   storage-class-attribute ::=

          AUTOMATIC | STATIC | CONTROLLED

(40)   defined-attribute ::=

          DEFINED basic-reference | POSITION ( expression )

(41)   based-attribute ::=

          BASED [ ( reference ) ]

(42)   initial-attribute ::=

          INITIAL { initial-call | initial-itemlist }

(43)   initial-call ::=

          CALL reference

(44)   initial-itemlist ::=

          ( { , • initial-item••• } )

(45)   initial-item ::=

          initial-iteration | initial-constant | simple-string-constant | reference |
          ( expression ) | *

(46)   initial-iteration ::=

          ( expression ) { initial-constant | initial-itemlist | reference }

(47)   initial-constant ::=

          replicated-string-constant | arithmetic-init-constant | sterling-constant

(48)   arithmetic-init-constant ::=

          [  + | - ] real-constant [ {  + | - } imaginary-constant ] |
          [  + | - ] imaginary-constant
```

(49) non-data-attribute ::=

BUILTIN | generic-attribute | attention-attribute

(50) entry-name-attribute ::=

ENTRY [(descriptorlist)] | returns-attribute | REDUCIBLE | IRREDUCIBLE

(51) descriptorlist ::=

descriptor [, descriptorlist]

(52) descriptor ::=

[integer] [dimension-attribute] [attribute•••] | *

(53) file-name-attribute ::=

FILE | file-attribute | ENVIRONMENT (env-option)

(54) file-attribute ::=

BITSTREAM | STREAM | RECORD | INPUT | OUTPUT | UPDATE | SEQUENTIAL | DIRECT |
BUFFERED | UNBUFFERED | KEYED | PRINT | BACKWARDS | EXCLUSIVE | TRANSIENT

(55) generic-attribute ::=

GENERIC ({ , • generic-element••• })

(56) generic-element ::=

reference WHEN (descriptorlist)

(57) scope-attribute ::=

INTERNAL | EXTERNAL

(58) like-attribute ::=

LIKE unsubscripted-reference

(59) attention-attribute ::=

ATTENTION ENVIRONMENT (env-option)

3.1.1.2 Formats

(60) format-sentence ::=

[prefixlist] labellist FORMAT formatlist ;

(61) formatlist ::=

({ , • format••• })

(62) format ::=

format-iteration | format-item

(63) format-iteration ::=

{ integer | (expression) } { format-item | formatlist }

(64) format-item ::=

data-format | control-format | remote-format

(65) data-format ::=

real-format | complex-format | string-format | picture-format

(66) real-format ::=

{ E | F } (expression [, expression [, expression]])

(67) complex-format ::=

C ({ real-format | picture-format } [, real-format | , picture-format])

(68) string-format ::=

{ A | B | BB } [(expression)]

(69) picture-format ::=

{ BP | P } picture-specification

(70) control-format ::=

{ BCOLUMN | BX | COLUMN | LINE | PAGE | SKIP | X } [(expression)]

(71) remote-format ::=

R (reference)

3.1.2 STATEMENTS

(72) statement ::=

[prefixlist] [labellist] { if-statement | unconditional-statement }

(73) prefixlist ::=

{ ({ , • prefix-element••• }) : }•••

(74) prefix-element ::=

prefix | no-prefix | check-condition | no-check-condition

(75) prefix ::=

CONVERSION | FIXEDOVERFLOW | OVERFLOW | STRINGRANGE | STRINGSIZE |
SUBSCRIPTRANGE | UNDERFLOW | ZERODIVIDE

(76) no-prefix ::=

NOCONVERSION | NOFIXEDOVERFLOW | NOOVERFLOW | NOSIZE | NOSTRINGSIZE |
NOSTRINGRANGE | NOSUBSCRIPTRANGE | NOUNDERFLOW | NOZERODIVIDE

(77) labellist ::=

{ basic-reference : }•••

(78) unconditional-statement ::=

begin-block | group | goto-statement | call-statement | incorporate-statement |
fetch-statement | release-statement | return-statement | wait-statement |
delay-statement | exit-statement | stop-statement | assignment-statement |
allocate-statement | free-statement | on-statement | revert-statement |
signal-statement | enable-statement | disable-statement | access-statement |
open-statement | close-statement | stream-io-statement | record-io-statement |
display-statement | null-statement

(79) null-statement ::=

;

3.1.2.1 Block and groups

(80) begin-block ::=

BEGIN [{ options-attribute | ORDER | REORDER }•••] ; sentencelist

(81) group ::=

simple-group | iterated-group

(82) simple-group ::=

DO ; sentencelist

(83) iterated-group ::=

DO { do-specification | WHILE (expression) } ; sentencelist

(84) do-specification ::=

reference = { , • specification••• }

(85) specification ::=

expression [BY expression [TO expression] |
TO expression [BY expression]] [WHILE (expression)]

3.1.2.2 Flow of control statements

(86) if-statement ::=

if-clause statement | if-clause balanced-statement ELSE statement

(87) if-clause ::=

IF expression THEN

(88) balanced-statement ::=

[prefixlist] [labellist] { if-clause balanced-statement ELSE
balanced-statement | unconditional-statement }

(89) goto-statement ::=

{ GOTO | GO TO } reference ;

(90) call-statement ::=

CALL reference [call-optionslist] ;

(91) call-optionslist ::=

{ TASK [(reference)] | PRIORITY (expression) | EVENT (reference) }•••

(92) return-statement ::=

RETURN [(expression)] ;

(93) incorporate-statement ::=

INCORPORATE (incorporate-specification)

(94) fetch-statement ::=

FETCH { , • reference••• }

(95) release-statement ::=

RELEASE { , • reference••• }

(96) wait-statement ::=

WAIT ({ , • reference••• }) [(expression)] ;

```
(97)   delay-statement ::=

          DELAY ( expression ) ;

(98)   exit-statement ::=

          EXIT ;

(99)   stop-statement ::=

          STOP ;
```

3.1.2.3 Storage manipulating statements

```
(100)  assignment-statement ::=

          { , * reference*** }  =  expression [ , BY NAME ] ;

(101)  allocate-statement ::=

          ALLOCATE { , * { based-allocate-item | controlled-allocate-item }*** } ;

(102)  based-allocate-item ::=

          identifier { SET ( reference ) [ IN ( reference ) ] |
          IN ( reference ) [ SET ( reference ) ] }

(103)  controlled-allocate-item ::=

          [ integer ] identifier [ dimension-attribute ] [ { string-attribute |
          area-attribute | initial-attribute }*** ]

(104)  free-statement ::=

          FREE { , * { reference [ IN ( reference ) ] }*** } ;
```

3.1.2.4 Condition and attention handling statements

```
(105)  on-statement ::=

          ON condition [ SNAP ] { unconditional-statement | SYSTEM ; }

(106)  revert-statement ::=

          REVERT condition ;

(107)  signal-statement ::=

          SIGNAL condition ;
```

(108) condition ::=

prefix | check-condition | AREA | named-io-condition | ERROR | FINISH |
programmer-named-condition | attention-condition

(109) check-condition ::=

CHECK ({ , • unsubscripted-reference••• })

(110) no-check-condition ::=

NOCHECK ({ , • unsubscripted-reference••• })

(111) named-io-condition ::=

io-condition (reference)

(112) io-condition ::=

BEGINVOLUME | ENDFILE | ENDPAGE | ENDVOLUME | KEY | NAME | PENDING | RECORD |
TRANSMIT | UNDEFINEDFILE

(113) programmer-named-condition ::=

CONDITION (identifier)

(114) attention-condition ::=

ATTENTION ({ , • identifier••• })

(115) access-statement ::=

ACCESS ATTENTION [({ , • identifier••• })] { ELSE statement | ; }

(116) enable-statement ::=

ENABLE { , • { attention-condition [ACCESS | ASYNC |
EVENT (reference)]••• }••• }

(117) disable-statement ::=

DISABLE attention-condition

3.1.2.5 Input and output statements

(118) open-statement ::=

OPEN { , • open-optionslist••• } ;

(119) open-optionslist ::=

{ file-attribute | FILE (reference) | BLINESIZE (expression) |
LINESIZE (expression) | PAGESIZE (expression) | TITLE (expression) |
ENVIRONMENT (env-option) | VOLUME }•••

```
(120)  close-statement ::=

          CLOSE { , • close-optionslist••• } ;

(121)  close-optionslist ::=

          { FILE ( reference ) | ENVIRONMENT ( env-option ) | VOLUME }•••

(122)  stream-io-statement ::=

          { GET | PUT } stream-optionslist ;

(123)  stream-optionslist ::=

          { FILE ( reference ) | BITSTRING ( expression ) | STRING ( expression ) |
          data-specification | COPY | SKIP [ ( expression ) ] | PAGE |
          LINE ( expression ) }•••

(124)  data-specification ::=

          data-directed | edit-directed | list-directed

(125)  data-directed ::=

          DATA [ ( datalist ) ]

(126)  edit-directed ::=

          EDIT { ( datalist ) formatlist }•••

(127)  list-directed ::=

          LIST ( datalist )

(128)  datalist ::=

          { , • datalist-element••• }

(129)  datalist-element ::=

          ( datalist DO do-specification ) | expression

(130)  record-io-statement ::=

          { READ | WRITE | REWRITE | LOCATE identifier | DELETE |
          UNLOCK } record-optionslist ;

(131)  record-optionslist ::=

          { FILE ( reference ) | EVENT ( reference ) | FROM ( reference ) |
          IGNORE ( expression ) | INTO ( reference ) | KEY ( expression ) |
          KEYTO ( reference ) | KEYFROM ( expression ) | SET ( reference ) | NOLOCK }•••
```

(132) display-statement ::=

DISPLAY (expression) [REPLY (reference) [EVENT (reference)] |
EVENT (reference) REPLY (reference)] ;

3.1.3 EXPRESSIONS

(133) expression ::=

expression-six | expression | expression-six

(134) expression-six ::=

expression-five | expression-six & expression-five

(135) expression-five ::=

expression-four | expression-five comparison-operator expression-four

(136) comparison-operator ::=

> | >= | = | < | <= | ¬> | ¬= | ¬<

(137) expression-four ::=

expression-three | expression-four || expression-three

(138) expression-three ::=

expression-two | expression-three { + | - } expression-two

(139) expression-two ::=

expression-one | expression-two { * | / } expression-one

(140) expression-one ::=

primitive-expression | { + | - | ¬ } expression-one |
primitive-expression ** expression-one

(141) primitive-expression ::=

(expression) | reference | constant | isub

(142) reference ::=

[reference ->] basic-reference

(143) basic-reference ::=

[{ identifier [subscriptlist] . }•••] identifier [subscriptlist•••]

(144) subscriptlist ::=

([, • { expression | * }•••])

(145) unsubscripted-reference ::=

[. • identifier•••]

(146) constant ::=

real-constant | imaginary-constant | sterling-constant |
simple-string-constant | replicated-string-constant

(147) replicated-string-constant ::=

(integer) simple-string-constant

3.2 LOWER LEVEL PRODUCTIONS

3.2.1 IDENTIFIERS AND CONSTANTS

(148) identifier ::=

letter [alphameric-character•••]

(149) letter ::=

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
P | Q | R | S | T | U | V | W | X | Y | Z | $ | @ | #

(150) alphameric-character ::=

letter | digit | _

(151) digit ::=

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

(152) isub ::=

integer SUB

(153) integer ::=

digit•••

(154) real-constant ::=

{ fixed-constant | float-constant } [B]

(155) fixed-constant ::=

integer [.] | [integer] . integer

(156) float-constant ::=

fixed-constant E [+ | -] integer

(157) imaginary-constant ::=

real-constant I

(158) simple-string-constant ::=

bit-string | character-string

(159) bit-string ::=

' [bit...] ' B

(160) bit ::=

0 | 1

(161) character-string ::=

' [string-character...] '

(162) string-character ::=

alphameric-character | BLANK | '' | = | + | - | * | / |
(|) | , | . | ; | : | & | ⊥ | ¬ | > | < | ? | % | extralingual-character

(163) sterling-constant ::=

integer . integer . fixed-constant L

3.2.2 PICTURES

(164) picture-specification ::=

' picture-string [F ([+ | -] integer)] '

(165) picture-string ::=

[(integer)] picture-character [picture-string]

(166) picture-character ::=

A | B | C | D | E | G | H | I | K | M | P | R | S | T | V |
X | Y | Z | $ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | + | - | * | / | , | .

3.2.3 BLANKS AND COMMENTS

```
(167)  space ::=

          { BLANK | comment }•••

(168)  comment ::=

          /  * [  { [ *••• ] comment-symbol | /  }••• ] *•••  /

(169)  comment-symbol ::=

          alphameric-character | BLANK | ' | = | + | - | ( | ) | , |
          . | ; | : | & | ⊥ | ¬ | > | < | ? | % | extralingual-character
```

3.3 LIST OF PL/I WORDS:

.
<
<=
(
+
|
||
&
*
**
)
;
¬
¬<
¬>
¬=
-
->
/
,
>
>=
:
=
A
ACCESS
ALIGNED
ALL
ALLOCATE
AREA
ASYNC
ATTENTION
AUTOMATIC
B
BB
BACKWARDS
BASED
BCOLUMN
BEGIN
BEGINVOLUME
BINARY
BIT
BITSTREAM
BITSTRING
BLINESIZE
BP
BUFFERED
BUILTIN
BX
BY
C
CALL
CHARACTER
CHECK
CLOSE
COLUMN
COMPLEX
CONDITION
CONNECTED
CONTROLLED
CONVERSION
COPY
DATA
DECIMAL
DECLARE
DEFAULT
DEFINED
DELAY
DELETE
DESCRIPTORS
DIRECT
DISABLE
DISPLAY
DO
E
EDIT
ELSE
ENABLE
END
ENDFILE
ENDPAGE
ENDVOLUME
ENTRY
ENVIRONMENT
ERROR
EVENT
EXCLUSIVE
EXIT
EXTERNAL
F
FETCH
FILE
FINISH
FIXED
FIXEDOVERFLOW
FLOAT
FORMAT
FREE
FROM
GENERIC
GET
GO
GOTO
IDENT
identifier
IF
IGNORE
imaginary-constant
IN
INCORPORATE
INITIAL
INPUT
integer
INTERNAL
INTO
IRREDUCIBLE
isub
KEY
KEYED
KEYFROM
KEYTO
LABEL
LIKE
LINE
LINESIZE
LIST
LOCATE
NAME
NOCHECK
NOCONVERSION
NOFIXEDOVERFLOW
NOLOCK
NOOVERFLOW
NOSIZE
NOSTRINGRANGE
NOSTRINGSIZE
NOSUBCRIPTRANGE
NOUNDERFLOW
NOZERODIVIDE
OFFSET
ON
OPEN
OPTIONS
ORDER
OUTPUT
OVERFLOW
P
PAGE
PAGESIZE
PENDING
PICTURE
picture-specification
POINTER
POSITION
PRINT
PRIORITY
PROCEDURE
PUT
R
RANGE
READ
REAL
real-constant
RECORD
RECURSIVE
REDUCIBLE
REFER
REORDER
RELEASE
REPLY
RETURN
RETURNS
REVERT
REWRITE
SECONDARY
SEQUENTIAL
SET
SIGNAL
simple-string-constant
SIZE
SKIP
SNAP
STATIC
sterling-constant
STOP
STREAM
STRING
STRINGRANGE
STRINGSIZE
SUBSCRIPTRANGE
SYSTEM
TASK
THEN
TITLE
TO
TRANSIENT
TRANSMIT
UNALIGNED
UNBUFFERED
UNDEFINEDFILE
UNDERFLOW
UNLOCK
UPDATE
VALUE
VARIABLE
VARYING
VOLUME
WAIT
WHEN
WHILE
WRITE
X
ZERODIVIDE

APPENDIX: CROSS-REFERENCE INDEX

This index lists all non-terminals and terminals of the productions of chapter 3.

A reference is given by the form 3-YY(ZZ), where YY is the page number within the chapter 3. For non-terminals the defining formula is indicated by an underlining.