CONCRETE SYNTAX OF PL/I

G. URSCHLER
NOTE

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.
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/ ALBEE, K., OLIVA, P., URSCHLIER, G.: Concrete Syntax of PL/I.

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

III).


/5/ WALK, K., ALBEE, K., TIECK, M., GOLLMANN, H., LAUER, P., MOSER, E., OLIVA, P.,
STIGLEITNER, H., ZEISL, G.: Abstract Syntax and Interpretation of PL/I (ULL
Version III).
IBM Laboratory Vienna, Techn. Report TR 25.098, 30 April 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/ ALBEE, K.: Syntactical Description of PL/I Text and its Translation into Abstract
Formal Form.

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.

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.

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, Würsley.
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. URSCHEL (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 ........................................... 1

2. SYNTAX NOTATION ........................................ 1
   2.1 Semantics of the Extended Backus Notation ............... 1
   2.2 The Syntax of the Extended Backus Notation ............... 2
   2.3 Generation of a Concrete Program Text .................. 3
       2.3.1 The normal generation process .................. 3
       2.3.2 Auxiliary rules for additional facilities ......... 5
           2.3.2.1 Keyword abbreviations ..................... 5
           2.3.2.2 Multiple closure of blocks and groups ....... 6
       2.3.3 Programs in the 48 character set ................ 6

3. CONCRETE SYNTAX ....................................... 1
   3.1 Higher Level Productions ............................ 1
       3.1.1 Declarations ................................ 1
           3.1.1.1 Attributes ............................ 3
           3.1.1.2 Formats ............................. 5
       3.1.2 Statements .................................. 6
           3.1.2.1 Block and groups ....................... 7
           3.1.2.2 Flow of control statements ............... 8
           3.1.2.3 Storage manipulating statements ............ 9
           3.1.2.4 Condition and attention handling statements 9
           3.1.2.5 Input and output statements ............... 10
       3.1.3 Expressions .................................. 12
   3.2 Lower Level Productions ............................ 13
       3.2.1 Identifiers and constants ....................... 13
       3.2.2 Pictures ................................... 14
       3.2.3 Blanks and comments .......................... 15
   3.3 List of PL/I Words: ................................ 16

APPENDIX: CROSS-REFERENCE INDEX
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.
2.1 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:

\[ V ::= S_1 | S_2 | \ldots | S_n \]

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

Note: In this chapter with \(V\) variables, with \(S_n\) (\(n \in \{1,2,3,\ldots\}\)) 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
\[ V ::= S_1 T_1 S_2 | S_1 T_2 S_2 | \ldots | S_1 T_n S_2 \]
may be replaced by
\[ V ::= S_1 \{ T_1 | T_2 | \ldots | T_n \} S_2 \]
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
\[ V ::= S_1 S_2 | S_1 T_1 S_2 | \ldots | S_1 T_n S_2 \]
may be replaced by
\[ V ::= S_1 \{ T_1 | T_2 | \ldots | T_n \} S_2 \]
and vice versa.
A production like

\[
\text{integer ::= digit | integer digit}
\]

may be shortened to

\[
\text{integer ::= digit}^{*}\]

In general

\[
V ::= U \mid V U \text{ or } V ::= U \mid U V
\]

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 \(V ::= U \mid U V\) or \(V ::= U \mid V U\) is to be added to the other production rules.

A production like

\[
\text{declarationlist ::= declaration [ } , \text{ declaration } ]^{*}\]

may be shortened to

\[
\text{declarationlist ::= } [, \text{ * declaration}^{*}\]

In general

\[
V ::= S_1 T_1 [ \{ T_2 T_3 \}^{*}\ ] S_2
\]

may be replaced by

\[
V ::= S_1 \{ T_2 \text{ * } T_3^{*}\ } S_2 \text{ and vice versa.}
\]

Note: Instead of \(S_1 [ \{ T_2 \text{ * } T_3^{*}\ } S_2\) also \(S_1 [ T_2 \text{ * } T_3^{*}\ ] S_2\) may be written.

### 2.2. 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 ':','=' and '['] while the or-sign of the PL/I production rules is denoted by ':'.

2.2 SYNTAX NOTATION
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 |

Auxiliary rules for the insertion of spaces:

Spaces (e.g., blanks, new lines, new pages) are optional immediately
preceding or succeeding "**" 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 69 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:

<table>
<thead>
<tr>
<th>keywords</th>
<th>abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTENTION</td>
<td>ATTN</td>
</tr>
<tr>
<td>AUTOMATIC</td>
<td>AUTO</td>
</tr>
<tr>
<td>BCOLUNW</td>
<td>BCOL</td>
</tr>
<tr>
<td>BEGINVOLUME</td>
<td>BOV</td>
</tr>
<tr>
<td>BINARY</td>
<td>BIN</td>
</tr>
<tr>
<td>BUFFERED</td>
<td>BUF</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>CHAR</td>
</tr>
<tr>
<td>COLUMN</td>
<td>COL</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>CPLX</td>
</tr>
<tr>
<td>CONNECTED</td>
<td>CONN</td>
</tr>
<tr>
<td>CONTROLLED</td>
<td>CTL</td>
</tr>
<tr>
<td>CONVERSION</td>
<td>CONV</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DEC</td>
</tr>
<tr>
<td>DECLARE</td>
<td>DCL</td>
</tr>
<tr>
<td>DEFINED</td>
<td>DEF</td>
</tr>
<tr>
<td>ENDOVOLUME</td>
<td>EOF</td>
</tr>
<tr>
<td>ENVIRONMENT</td>
<td>ENV</td>
</tr>
<tr>
<td>EXCLUSIVE</td>
<td>EXCL</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>EXT</td>
</tr>
<tr>
<td>FIXEDOVERFLOW</td>
<td>FOFL</td>
</tr>
<tr>
<td>INITIAL</td>
<td>INIT</td>
</tr>
<tr>
<td>INTERNAL</td>
<td>INT</td>
</tr>
<tr>
<td>IRREDUCIBLE</td>
<td>IBERD</td>
</tr>
<tr>
<td>NOCONVERSION</td>
<td>NOCONV</td>
</tr>
<tr>
<td>NOPROCEDFLOW</td>
<td>NOPROF</td>
</tr>
<tr>
<td>NOOVERFLOW</td>
<td>NOOFL</td>
</tr>
<tr>
<td>NOTSTRINGRANGE</td>
<td>NOTSTRG</td>
</tr>
<tr>
<td>NOUNDEFINETIME</td>
<td>NOUNDEF</td>
</tr>
<tr>
<td>NOUNDEFINDATE</td>
<td>NOUNDE</td>
</tr>
<tr>
<td>NOZERODIVIDE</td>
<td>NOZDIV</td>
</tr>
<tr>
<td>OVERFLOW</td>
<td>OFL</td>
</tr>
<tr>
<td>PICTURE</td>
<td>PIC</td>
</tr>
<tr>
<td>POINTER</td>
<td>PTR</td>
</tr>
<tr>
<td>POSITION</td>
<td>POS</td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>PROC</td>
</tr>
<tr>
<td>REDUCIBLE</td>
<td>RED</td>
</tr>
<tr>
<td>SEQUENTIAL</td>
<td>SEQL</td>
</tr>
<tr>
<td>STRINGRANGE</td>
<td>STRG</td>
</tr>
<tr>
<td>STRINGSIZE</td>
<td>STRZ</td>
</tr>
<tr>
<td>SUBSTRINGRANGE</td>
<td>SUBSTRG</td>
</tr>
<tr>
<td>UNALIGNED</td>
<td>UNAL</td>
</tr>
<tr>
<td>UNBUFFERED</td>
<td>UNBFSF</td>
</tr>
<tr>
<td>UNDEFINEDFILE</td>
<td>UNDENT</td>
</tr>
<tr>
<td>UNDERFLOW</td>
<td>UFL</td>
</tr>
<tr>
<td>VARYING</td>
<td>VAR</td>
</tr>
<tr>
<td>ZERODIVIDE</td>
<td>ZDIV</td>
</tr>
</tbody>
</table>

2. SYNTAX NOTATION 5
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:

\[
\text{compound} ::= \text{procedure} \mid [\text{prefixlist}] [\text{labellist}] \begin{align*}& \mid \text{begin-block} \mid \text{group}\end{align*}
\]

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:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789abcdefghijklmnopqrstuvwxyz
- + * ( ) : ,
```

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", "alphamer-" and "string-character" and "comment-symbol" the following 12 symbols have to be deleted:

`? %`<n>

(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
: ::= ::= LE
& ::= ::= HG
| ::= ::= NL
> ::= ::= ME
< ::= ::= 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


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

\[
; ::= \text{space} .
\]

(b) else by means of the production rule

\[
; ::= .
\]

6 2. SYNTAX NOTATION
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 ;
CONCRETE SYNTAX OF PL/I

30 June 1969

(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 | system

(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
3.1.1.1 attributes

options-attribute ::=  
   OPTIONS ( [ , * external-option*** ] )

returns-attribute ::=  
   RETURNS ( [ data-attribute | entry-name-attribute | FILE ]*** )

dimension-attribute ::=  
   ( [ , * bound-pair*** ] )

bound-pair ::=  
   [ refer-expression : ] refer-expression | *

refer-expression ::=  
   expression [ REFER ( unsubscripted-reference ) ]

attribute ::=  
   data-attribute | non-data-attribute | entry-name-attribute |  
   file-name-attribute | scope-attribute | like-attribute

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

arithmetic-attribute ::=  
   [ REAL | COMPLEX | DECIMAL | BINARY | FLOAT |  
   FIXED ] [ { integer [ , signed-integer ] } ]

signed-integer ::=  
   [ + ] - ] integer

string-attribute ::=  
   [ BIT | CHARACTER ] [ ( { refer-expression | * } ) ]

picture-attribute ::=  
   PICTURE [ picture-specification ]
area-attribute ::= AREA [ ( ( refer-expression * ) ) ]

label-attribute ::= LABEL [ ( , * identifier ) ]

offset-attribute ::= OFFSET [ ( reference ) ]

storage-class-attribute ::= AUTOMATIC | STATIC | CONTROLLED

defined-attribute ::= DEFINED basic-reference | POSITION ( expression )

based-attribute ::= BASED [ ( reference ) ]

initial-attribute ::= INITIAL [ initial-call | initial-itemlist ]

initial-call ::= CALL reference

initial-itemlist ::= ( [ , * initial-item ] )

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

initial-iteration ::= ( expression ) [ initial-constant | initial-itemlist | reference ]

initial-constant ::= replicated-string-constant | arithmetic-init-constant | sterling-constant

arithmetic-init-constant ::= [ + ] real-constant [ [ + ] - imaginary-constant ] | [ + ] - imaginary-constant

4 3. CONCRETE SYNTAX
CONCRETE SYNTAX OF PL/I

3.1.1.2 Formats

format-sentence ::=  
[ prefixlist ] labellist FORMAT formatlist ;

(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. CONCRETE SYNTAX 5
CONCRETE SYNTAX OF PL/I

3.1.2 STATEMENTS

(72)  statement ::=  
     [ prefixlist ] [ labellist ] [ if-statement | unconditional-statement ]

(73)  prefixlist ::=  
(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 | NOPFIXEDOVERFLOW | 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 ::=  
    ;

3.12 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*** ]
3.1.2.2 Flow of control statements

specification ::= expression [ BY expression [ TO expression ] ] [ TO expression [ BY expression ] ] [ WHILE ( expression ) ]

if-clause ::= IF expression THEN

balanced-statement ::= [ prefixlist ] [ labellist ] [ if-clause balanced-statement ELSE balanced-statement ] unconditional-statement

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

call-statement ::= CALL reference [ call-optionslist ] ;

call-optionslist ::= 
{ TASK [ ( reference ) ] | PRIORITY ( expression ) | EVENT ( reference ) }...

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

incorporate-statement ::= 
INCORPORATE ( incorporate-specification )

fetch-statement ::= 
FETCH [ , * reference • • • ]

release-statement ::= 
RELEASE [ , * reference • • • ]

wait-statement ::= 
WAIT [ ( , * reference • • • ) ] [ ( expression ) ] ;
CONCRETE SYNTAX OF PL/I

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 [ SWAP ] [ unconditional-statement | SYSTEM ; ]

(106) revert-statement ::= 
    REVERT condition ;

(107) signal-statement ::= 
    SIGNAL condition ;
CONCRETE SYNTAX OF PL/I

(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 | ENDPAGE | ENDPAGE | INDVOLUME | 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 ::= 
    \[ reference \rightarrow \] basic-reference

(143) basic-reference ::= 
    \[ \[ identifier [ subscriptlist ] \. ]*** \] identifier [ subscriptlist*** ]

12 3. CONCRETE SYNTAX
subscriptlist ::=  
    ( [ , * ( expression | * )••• ] )

un subscripted-reference ::=  
    [ . * identifier••• ]

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

replicated-string-constant ::=  
    ( integer ) simple-string-constant

3.2 LOWER-LEVEL PRODUCTIONS

3.2.1 IDENTIFIERS AND CONSTANTS

identifier ::=  
    letter [ alphabetic-character••• ]

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 | $ | @ | #

alphabetic-character ::=  
    letter | digit | _

digit ::=  
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

isub ::=  
    integer SUB

integer ::=  
    digit•••

real-constant ::=  
    [ fixed-constant | float-constant ] [ B ]

fixed-constant ::=  
    integer [ . ] [ integer ] . integer
float-constant ::= fixed-constant E [+ | - ] integer

imaginary-constant ::= real-constant I

simple-string-constant ::= bit-string | character-string

bit-string ::= ' [ bit••• ] ' B

bit ::= 0 | 1

character-string ::= ' [ string-character••• ] '

string-character ::= alphabetic-character | BLANK | "" | = | + | - | * | / | { | } | ] | ; | : | : | & | [ | ] | ~ | > | < | ? | % | extralingual-character

sterling-constant ::= integer . integer . fixed-constant L

4.2.2 PICTURES

3.2.3 BLANKS AND COMMENTS

(167) space ::=  
    { BLANK | comment }***

(168) comment ::=  
    / * [ { [ **** ] comment-symbol | / }*** ] *** /

(169) comment-symbol ::=  
    alphabetic-character | BLANK | ' | ] | + | - | ( | ) | : |  
    . | ; | | & | | ~ | | > | | < | | ? | | extralingual-character
### 3.3 List of PL/I Words:

<table>
<thead>
<tr>
<th>Word</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK</td>
<td>Imaginary-constant</td>
</tr>
<tr>
<td>BoxFit</td>
<td>In</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>Incorporate</td>
</tr>
<tr>
<td>CONDITION</td>
<td>Initial</td>
</tr>
<tr>
<td>CONNECTED</td>
<td>Input</td>
</tr>
<tr>
<td>CONTROLLED</td>
<td>Integer</td>
</tr>
<tr>
<td>CONVERSION</td>
<td>Internal</td>
</tr>
<tr>
<td>COPY</td>
<td>Into</td>
</tr>
<tr>
<td>DECLARE</td>
<td>Isub</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>Key</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Keyed</td>
</tr>
<tr>
<td>DEFINED</td>
<td>KeyFrom</td>
</tr>
<tr>
<td>DELAY</td>
<td>Keyto</td>
</tr>
<tr>
<td>DESCRIBERS</td>
<td>Like</td>
</tr>
<tr>
<td>DIRECT</td>
<td>Line</td>
</tr>
<tr>
<td>DISABLE</td>
<td>LINESIZE</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>LIST</td>
</tr>
<tr>
<td>DO</td>
<td>LOCATE</td>
</tr>
<tr>
<td>ELSE</td>
<td>NAME</td>
</tr>
<tr>
<td>EDIT</td>
<td>NOCHECK</td>
</tr>
<tr>
<td>ELSE</td>
<td>NOCONVERSION</td>
</tr>
<tr>
<td>ENABLE</td>
<td>NOPREFIXOVERFLOW</td>
</tr>
<tr>
<td>ACCESS</td>
<td>NOPREFIXOVERFLOW</td>
</tr>
<tr>
<td>ALIGNED</td>
<td>NOSIZE</td>
</tr>
<tr>
<td>ALL</td>
<td>NOUNDERFLOW</td>
</tr>
<tr>
<td>ALLOCATE</td>
<td>NOZEROOVERFLOW</td>
</tr>
<tr>
<td>AREA</td>
<td>STERLING-CONSTANT</td>
</tr>
<tr>
<td>ASYNC</td>
<td>STOP</td>
</tr>
<tr>
<td>ATTENTION</td>
<td>STREAM</td>
</tr>
<tr>
<td>AUTOMATIC</td>
<td>STRING</td>
</tr>
<tr>
<td>B</td>
<td>SUBSCRIPTRANGE</td>
</tr>
<tr>
<td>BASED</td>
<td>SUBSTRING</td>
</tr>
<tr>
<td>BEGIN</td>
<td>STRINGSIZE</td>
</tr>
<tr>
<td>BEGINVOLUME</td>
<td>SYSTEM</td>
</tr>
<tr>
<td>BINARY</td>
<td>TASK</td>
</tr>
<tr>
<td>BIT</td>
<td>TITLE</td>
</tr>
<tr>
<td>BITSTREAM</td>
<td>TO</td>
</tr>
<tr>
<td>BITSTRING</td>
<td>TRANSIENT</td>
</tr>
<tr>
<td>BLINESIZE</td>
<td>TRANSIENT</td>
</tr>
<tr>
<td>BOMET</td>
<td>UNALIGNED</td>
</tr>
<tr>
<td>BUFFERED</td>
<td>UNBUFFERED</td>
</tr>
<tr>
<td>BUILTIN</td>
<td>UNDEFINEDFILE</td>
</tr>
<tr>
<td>BX</td>
<td>UNDERFLOW</td>
</tr>
<tr>
<td>BY</td>
<td>UNLOCK</td>
</tr>
<tr>
<td>CALL</td>
<td>UPDATE</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>VALUE</td>
</tr>
<tr>
<td>C</td>
<td>VARIABLE</td>
</tr>
<tr>
<td>CALL</td>
<td>ZERODIVIDE</td>
</tr>
<tr>
<td>CHAR</td>
<td>ZERODIVIDE</td>
</tr>
<tr>
<td>IDENTIFIER</td>
<td></td>
</tr>
</tbody>
</table>

---

3. CONCRETE SYNTAX
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.
CONCRETE SYNTAX OF PL/I

30 June 1969

APPENDIX: CROSS-REFERENCE INDEX
<table>
<thead>
<tr>
<th>Keywords</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>based-allocate-item</td>
<td>3-9 (102), 3-9 (101)</td>
</tr>
<tr>
<td>based-attribute</td>
<td>3-9 (121), 3-3 (31)</td>
</tr>
<tr>
<td>basic-reference</td>
<td>3-1 (136), 3-9 (40), 3-7 (77), 3-12 (142)</td>
</tr>
<tr>
<td>BB</td>
<td>3-6 (69)</td>
</tr>
<tr>
<td>COLUMN</td>
<td>3-6 (70)</td>
</tr>
<tr>
<td>BEGIN</td>
<td>3-7 (80)</td>
</tr>
<tr>
<td>begin-block</td>
<td>3-7 (169), 3-7 (78)</td>
</tr>
<tr>
<td>BEGINVOLUME</td>
<td>3-10 (112)</td>
</tr>
<tr>
<td>BITNARY</td>
<td>3-3 (32)</td>
</tr>
<tr>
<td>BIT</td>
<td>3-14 (160), 3-14 (159)</td>
</tr>
<tr>
<td>BITT</td>
<td>3-3 (34)</td>
</tr>
<tr>
<td>BITSTRING</td>
<td>3-14 (159), 3-14 (158)</td>
</tr>
<tr>
<td>BITSTREAM</td>
<td>3-5 (34)</td>
</tr>
<tr>
<td>BITSTRING</td>
<td>3-11 (123)</td>
</tr>
<tr>
<td>BLANK</td>
<td>3-14 (162), 3-15 (167), 3-15 (169)</td>
</tr>
<tr>
<td>BLINSETIZE</td>
<td>3-10 (119)</td>
</tr>
<tr>
<td>bound-pair</td>
<td>3-3 (28), 3-3 (27)</td>
</tr>
<tr>
<td>BR</td>
<td>3-6 (69)</td>
</tr>
<tr>
<td>BUFFERED</td>
<td>3-5 (54)</td>
</tr>
<tr>
<td>BULLETIN</td>
<td>3-5 (89)</td>
</tr>
<tr>
<td>BY</td>
<td>3-6 (70)</td>
</tr>
<tr>
<td>BY</td>
<td>3-8 (85), 3-9 (100)</td>
</tr>
<tr>
<td>C</td>
<td>3-6 (67), 3-13 (149), 3-14 (166)</td>
</tr>
<tr>
<td>CALL</td>
<td>3-4 (43), 3-8 (90)</td>
</tr>
<tr>
<td>call-optionslist</td>
<td>3-9 (121), 3-8 (90)</td>
</tr>
<tr>
<td>call-statement</td>
<td>3-3 (190), 3-7 (78)</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>3-3 (34)</td>
</tr>
<tr>
<td>character-string</td>
<td>3-14 (161), 3-14 (158)</td>
</tr>
<tr>
<td>CHECK</td>
<td>3-10 (109)</td>
</tr>
<tr>
<td>check-condition</td>
<td>3-10 (109), 3-7 (74), 3-10 (109)</td>
</tr>
<tr>
<td>CLOSE</td>
<td>3-11 (120)</td>
</tr>
<tr>
<td>close-optionslist</td>
<td>3-11 (121), 3-11 (120)</td>
</tr>
<tr>
<td>close-statement</td>
<td>3-11 (129), 3-7 (78)</td>
</tr>
<tr>
<td>COLUMN</td>
<td>3-6 (70)</td>
</tr>
</tbody>
</table>
APPENDIX: CROSS-REFERENCE INDEX
defined-attribute ........................................ 3-4(40), 3-3(31)
DELAY ......................................................... 3-9(97)
delay-statement ............................................. 3-9(97), 3-7(78)
DELETE .......................................................... 3-11(130)
descriptor ...................................................... 3-5(51), 3-5(51)
descriptorlist ................................................ 3-5(51), 3-5(56)
DESCRIPTOPS .................................................. 3-2(18)
digit ............................................................... 3-13(151), 3-13(150), 3-13(153)
dimension-attribute ......................................... 3-3(27), 3-2(12), 3-2(20), 3-5(52), 3-9(103)
DIRECT ............................................................ 3-5(54)
DISABLE .......................................................... 3-10(117)
disable-statement ............................................. 3-10(117), 3-7(78)
DISPLAY .......................................................... 3-12(132)
display-statement ............................................. 3-12(132), 3-7(78)
DO ................................................................. 3-7(82), 3-7(83), 3-11(129)
do-specification .............................................. 3-7(84), 3-7(83), 3-11(129)
E ................................................................. 3-6(66), 3-13(149), 3-14(156), 3-14(166)
EDIT .............................................................. 3-11(126)
edit-directed .................................................. 3-11(126), 3-11(124)
PLSE ............................................................. 3-8(86), 3-8(88), 3-10(115)
ENABLE .......................................................... 3-10(116)
enable-statement ............................................. 3-10(116), 3-7(78)
END .............................................................. 3-1(6)
end-clause ..................................................... 3-1(6), 3-1(5)
ENDFILE ........................................................ 3-10(112)
ENDPAGE ........................................................ 3-10(112)
ENDVOLUME ..................................................... 3-10(112)
entry ............................................................. 3-1(8), 3-1(7)
ENTRY ............................................................. 3-1(8), 3-5(50)
entry-name-attribute ....................................... 3-5(50), 3-3(26), 3-3(30)
en-environment .............................................. 3-5(53), 3-5(59), 3-10(119), 3-11(121)
environment .................................................. 3-5(53), 3-5(59), 3-10(119), 3-11(121)
ERROR .......................................................... 3-10(108)
EVENT .......................................................... 3-3(31), 3-3(31), 3-10(116), 3-11(131), 3-12(132)
<table>
<thead>
<tr>
<th>Definition</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCLUSIVE</td>
<td>3-5 (54)</td>
</tr>
<tr>
<td>EXIT</td>
<td></td>
</tr>
<tr>
<td>exit-statement</td>
<td>3-9 (98)</td>
</tr>
<tr>
<td>expression</td>
<td>3-12(135), 3-12(134), 3-12(135)</td>
</tr>
<tr>
<td>expression-five</td>
<td>3-12(135), 3-12(134), 3-12(135)</td>
</tr>
<tr>
<td>expression-four</td>
<td>3-12(135), 3-12(134), 3-12(135)</td>
</tr>
<tr>
<td>expression-one</td>
<td>3-12(139), 3-12(133), 3-12(134)</td>
</tr>
<tr>
<td>expression-six</td>
<td>3-12(139), 3-12(133), 3-12(134)</td>
</tr>
<tr>
<td>expression-three</td>
<td>3-12(139), 3-12(133), 3-12(134)</td>
</tr>
<tr>
<td>expression-two</td>
<td>3-12(139), 3-12(133), 3-12(134)</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>external-option</td>
<td></td>
</tr>
<tr>
<td>extralingual-character</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>factored-default-spec</td>
<td></td>
</tr>
<tr>
<td>FETCH</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>fetch-statement</td>
<td></td>
</tr>
<tr>
<td>FILE</td>
<td>3-3 (26), 3-5 (53), 3-7 (75)</td>
</tr>
<tr>
<td>file-attribute</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>file-name-attribute</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>FINISH</td>
<td></td>
</tr>
<tr>
<td>FIXED</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>fixed-constant</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>FIXEDOVERFLOW</td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>float-constant</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>format</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>FORMAT</td>
<td></td>
</tr>
<tr>
<td>format-item</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>format-iteration</td>
<td></td>
</tr>
<tr>
<td>format-sentence</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>formatlist</td>
<td>3-4 (40), 3-4 (45), 3-5 (63), 3-6 (66), 3-7 (70), 3-8 (85), 3-8 (87), 3-9 (91), 3-9 (92), 3-9 (96), 3-9 (97), 3-9 (100), 3-10 (119), 3-11 (123), 3-11 (129), 3-11 (131), 3-12 (133), 3-12 (134), 3-12 (135)</td>
</tr>
<tr>
<td>PRINT</td>
<td></td>
</tr>
</tbody>
</table>

APPENDIX: CROSS-REFERENCE INDEX
CONCRETE SYNTAX OF PL/I

free-statement ........................................ 3-9(104), 3-7(78)
FROM ....................................................... 3-11(131)
G .......................................................... 3-13(149), 3-14(166)
GENERIC .................................................. 3-5(55), 3-5(49)
generic-attribute ........................................ 3-5(56), 3-5(55)
generic-element .......................................... 3-11(122)
GET .......................................................... 3-8(89)
GO ...................................................................... 3-8(89)
GOTO .......................................................... 3-8(89)
goto-statement .............................................. 3-9(102), 3-7(78)
group ........................................................... 3-7(81), 3-7(78)
H ................................................................. 3-13(149), 3-14(166)
I ................................................................. 3-13(149), 3-14(157), 3-14(166)
identifier .................................................. 3-13(149), 3-1(3), 3-2(12), 3-2(19), 3-4(37), 3-9(102), 3-9(103), 3-10(113), 3-10(114), 3-10(115), 3-11(130), 3-12(143), 3-13(145)
identifier-range-spec ..................................... 3-2(119), 3-2(18)
IF ...................................................................... 3-8(87)
if-clause ........................................................ 3-8(87), 3-8(86), 3-8(89)
if-statement .................................................. 3-8(86), 3-6(72)
IGNORE .......................................................... 3-11(131)
imaginary-constant ....................................... 3-14(157), 3-4(48), 3-13(146)
IN ............................................................... 3-9(102), 3-9(104)
INTEGRATE .................................................... 3-8(93)
incorporate-specification ................................... 3-8(93)
incorporate-statement ..................................... 3-9(93), 3-7(78)
INITIAL .......................................................... 3-8(42)
initial-attribute .......................................... 3-3(31), 3-9(103)
initial-call .................................................... 3-9(43), 3-4(42)
initial-constant ............................................. 3-4(47), 3-4(45), 3-4(46)
initial-item .................................................... 3-4(51), 3-4(44)
initial-itemlist ............................................... 3-4(49), 3-4(42), 3-4(46)
initial-iteration .......................................... 3-4(46), 3-4(45)
INPUT ............................................................ 3-5(54)
integer ...................................................... 3-13(153), 3-2(12), 3-3(32), 3-3(33), 3-5(52), 3-6(63), 3-9(103), 3-13(147), 3-13(152), 3-13(155), 3-14(156), 3-14(163), 3-14(164), 3-14(165)

9 APPENDIX: CROSS-REFERENCE INDEX
<table>
<thead>
<tr>
<th>Term</th>
<th>Reference(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T3M</td>
<td></td>
</tr>
<tr>
<td>LAD</td>
<td></td>
</tr>
<tr>
<td>VIENNA</td>
<td></td>
</tr>
<tr>
<td>INTERNAL</td>
<td>3-5 (57)</td>
</tr>
<tr>
<td>INTO</td>
<td>3-11 (131)</td>
</tr>
<tr>
<td>io-condition</td>
<td>3-10 (112), 3-10 (111)</td>
</tr>
<tr>
<td>TRANSCODIBLE</td>
<td>3-9 (50)</td>
</tr>
<tr>
<td>isub</td>
<td>3-12 (114)</td>
</tr>
<tr>
<td>iterated-group</td>
<td>3-7 (89), 3-7 (81)</td>
</tr>
<tr>
<td>L</td>
<td>3-13 (119)</td>
</tr>
<tr>
<td>L LABEL</td>
<td>3-13 (119), 3-14 (163)</td>
</tr>
<tr>
<td>label-attribute</td>
<td>3-4 (37), 3-3 (31)</td>
</tr>
<tr>
<td>label-list</td>
<td>3-7 (271), 3-1 (2), 3-1 (6), 3-1 (8), 3-1 (9), 3-5 (60), 3-6 (72), 3-8 (88)</td>
</tr>
<tr>
<td>letter</td>
<td>3-13 (119), 3-2 (19), 3-13 (148), 3-13 (150)</td>
</tr>
<tr>
<td>LIKE</td>
<td>3-5 (58)</td>
</tr>
<tr>
<td>like-attribute</td>
<td>3-5 (58)</td>
</tr>
<tr>
<td>LINE</td>
<td>3-6 (70), 3-11 (123)</td>
</tr>
<tr>
<td>LINESIZE</td>
<td>3-10 (119)</td>
</tr>
<tr>
<td>LIST</td>
<td>3-11 (127)</td>
</tr>
<tr>
<td>list-directed</td>
<td>3-11 (127), 3-11 (124)</td>
</tr>
<tr>
<td>LOCATE</td>
<td>3-11 (130)</td>
</tr>
<tr>
<td>M</td>
<td>3-13 (149), 3-14 (165)</td>
</tr>
<tr>
<td>N</td>
<td>3-13 (149)</td>
</tr>
<tr>
<td>NAME</td>
<td>3-9 (100), 3-10 (112)</td>
</tr>
<tr>
<td>named-io-condition</td>
<td>3-10 (111), 3-10 (108)</td>
</tr>
<tr>
<td>no-check-condition</td>
<td>3-7 (74)</td>
</tr>
<tr>
<td>no-prefix</td>
<td>3-7 (74)</td>
</tr>
<tr>
<td>NOCHECK</td>
<td>3-10 (110)</td>
</tr>
<tr>
<td>NOCONVERSION</td>
<td>3-7 (76)</td>
</tr>
<tr>
<td>NOTTXYEDOVERFLOW</td>
<td>3-7 (76)</td>
</tr>
<tr>
<td>NOLOCK</td>
<td>3-11 (131)</td>
</tr>
</tbody>
</table>

APPENDIX: CROSS-REFERENCE INDEX 9
non-data-attribute ........................................ 3-5(43), 3-3(30)
NOCYCLEFLOW ................................................. 3-7(76)
NOSIZE ............................................................ 3-7(76)
NOSTRINGRANGE .............................................. 3-7(76)
NOSTRINGSIZE .................................................. 3-7(76)
NOSUBSCRIPTRANGE .......................................... 3-7(76)
NOWINDFLOW ..................................................... 3-7(76)
NOZERO_DIVIDE .................................................. 3-7(76)
null-statement ................................................. 3-7(79), 3-7(78)
O ................................................................. 3-13(149)
OFFSET ............................................................. 3-4(39)
offset-attribute .............................................. 3-4(38), 3-3(31)
ON ................................................................. 3-9(105)
on-statement ..................................................... 3-9(105), 3-7(78)
OPEN ............................................................... 3-10(118)
open-optionslist .............................................. 3-10(118)
open-statement ................................................. 3-10(118), 3-7(78)
OPTIONS ........................................................... 3-3(25)
options-attribute ........................................... 3-3(25), 3-1(4), 3-7(80)
ORDER ............................................................. 3-1(4), 3-7(80)
OUTPUT ............................................................. 3-5(54)
OVERFLOW ........................................................ 3-7(75)
P ................................................................. 3-6(69), 3-13(149), 3-14(166)
PAGE .............................................................. 3-6(70), 3-11(123)
PAGESIZE .......................................................... 3-10(119)
parameterlist .................................................... 3-1(3), 3-1(2), 3-1(8)
PENDING ........................................................... 3-10(112)
PICTURE ............................................................ 3-3(35)
picture-attribute ............................................. 3-3(25), 3-3(31)
picture-character ............................................. 3-14(165), 3-14(165)
picture-format .................................................. 3-6(69), 3-6(65), 3-6(67)
picture-specification ........................................ 3-14(164), 3-3(35), 3-6(69)
picture-string ............................................... 3-14(165), 3-14(164), 3-14(165)
POINTER ............................................................ 3-3(31)
CONCRETE SYNTAX OF PL/I

POSTITION ............................................. 3-4 (40)
precision-spec ........................................ 3-3 (29), 3-2 (23)
prefix ................................................... 3-7 (75), 3-7 (74), 3-10 (108)
prefix-element ........................................ 3-7 (74), 3-6 (73)
prefixlist .............................................. 3-6 (72), 3-8 (88)
primitive-expression ................................. 3-12 (141), 3-12 (140)
PRINT .................................................... 3-5 (54)
PRIORITY ................................................ 3-8 (91)
procedure ............................................. 3-1 (2), 3-1 (1), 3-1 (7)
PROCEDURE ............................................ 3-1 (2)
procedure-optionelist ............................... 3-1 (8), 3-1 (2)
program .............................................. 3-1 (1)
programmer-named-condition ...................... 3-10 (113), 3-10 (108)
PUT ...................................................... 3-11 (122)
Q .......................................................... 3-13 (149)
R .......................................................... 3-6 (71), 3-13 (149), 3-14 (166)
RANGE .................................................. 3-2 (19)
range-spec ............................................ 3-2 (18), 3-2 (17)
READ ..................................................... 3-11 (130)
REAL .................................................... 3-3 (32)
real-constant ........................................ 3-12 (154), 3-4 (48), 3-13 (146), 3-14 (157)
real-format ........................................... 3-6 (66), 3-6 (65), 3-6 (67)
RECORD .............................................. 3-5 (54), 3-10 (112)
record-io-statement ................................. 3-11 (130), 3-7 (78)
record-optionelist ................................. 3-11 (131), 3-11 (130)
RECURSIVE ............................................ 3-1 (4)
REDUCIBLE ............................................ 3-5 (50)
REFER .................................................. 3-3 (29)
reference .............................................. 3-12 (152), 3-4 (38), 3-4 (41), 3-4 (43), 3-4 (45), 3-4 (46), 3-5 (56), 3-6 (71), 3-7 (84),
3-8 (89), 3-8 (90), 3-8 (91), 3-8 (94), 3-8 (95), 3-8 (96), 3-9 (100), 3-9 (702), 3-9 (104),
3-10 (111), 3-10 (116), 3-10 (119), 3-11 (121), 3-11 (123), 3-11 (131), 3-12 (132), 3-12 (141),
3-12 (142)
RELEASE ............................................. 3-8 (95)
release-statement ................................. 3-8 (45), 3-7 (78)

APPENDIX: CROSS-REFERENCE INDEX
<table>
<thead>
<tr>
<th>SLANG</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE-FORMAT</td>
<td>3-6, 71, 3-6, 64</td>
</tr>
<tr>
<td>REORDER</td>
<td>3-1, 4, 3-7, 80</td>
</tr>
<tr>
<td>REPlicated-string-constant</td>
<td>3-13, 147, 3-4, 47, 3-13, 146</td>
</tr>
<tr>
<td>REPLY</td>
<td>3-12, 132</td>
</tr>
<tr>
<td>RETURN</td>
<td>3-8, 92</td>
</tr>
<tr>
<td>RETURN-statement</td>
<td>3-8, 92, 3-7, 78</td>
</tr>
<tr>
<td>RETURNS</td>
<td>3-3, 26</td>
</tr>
<tr>
<td>returns-attribute</td>
<td>3-3, 26, 3-1, 4, 3-1, 8, 3-5, 50</td>
</tr>
<tr>
<td>REVERT</td>
<td>3-9, 106</td>
</tr>
<tr>
<td>revert-statement</td>
<td>3-9, 106, 3-7, 78</td>
</tr>
<tr>
<td>RENAME</td>
<td>3-11, 130</td>
</tr>
<tr>
<td>SEND</td>
<td>3-13, 149, 3-14, 166</td>
</tr>
<tr>
<td>SCOPE-attribute</td>
<td>3-5, 57, 3-3, 30</td>
</tr>
<tr>
<td>SECONDARY</td>
<td>3-3, 31</td>
</tr>
<tr>
<td>SENTENCE</td>
<td>3-1, 17, 3-1, 5</td>
</tr>
<tr>
<td>SENTENCelist</td>
<td>3-1, 17, 3-1, 2, 3-7, 80, 3-7, 82, 3-7, 83</td>
</tr>
<tr>
<td>SEQUENTIAL</td>
<td>3-5, 54</td>
</tr>
<tr>
<td>SET</td>
<td>3-9, 102, 3-11, 131</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>3-9, 187</td>
</tr>
<tr>
<td>SIGNAL-statement</td>
<td>3-9, 107, 3-7, 78</td>
</tr>
<tr>
<td>SIGNED-integer</td>
<td>3-3, 33, 3-3, 32</td>
</tr>
<tr>
<td>SIMPLE-default-spec</td>
<td>3-2, 17, 3-2, 16</td>
</tr>
<tr>
<td>SIMPLE-group</td>
<td>3-7, 82, 3-7, 81</td>
</tr>
<tr>
<td>SIMPLE-string-constant</td>
<td>3-14, 158, 3-4, 45, 3-13, 146, 3-13, 147</td>
</tr>
<tr>
<td>SKIP</td>
<td>3-6, 70, 3-11, 123</td>
</tr>
<tr>
<td>SNAP</td>
<td>3-9, 105</td>
</tr>
<tr>
<td>SPACE</td>
<td>3-15, 167</td>
</tr>
<tr>
<td>SPECIFICATION</td>
<td>3-8, 95, 3-7, 84</td>
</tr>
<tr>
<td>STATEMENT</td>
<td>3-6, 72, 3-1, 7, 3-8, 86, 3-10, 115</td>
</tr>
<tr>
<td>STATIC</td>
<td>3-4, 39</td>
</tr>
<tr>
<td>STERLING-constant</td>
<td>3-14, 163, 3-4, 47, 3-13, 146</td>
</tr>
<tr>
<td>STOP</td>
<td>3-9, 99</td>
</tr>
<tr>
<td>STOP-statement</td>
<td>3-9, 99, 3-7, 78</td>
</tr>
<tr>
<td>STORAGE-class-attribute</td>
<td>3-9, 191, 3-7, 78</td>
</tr>
</tbody>
</table>

12 APPENDIX: CROSS-REFERENCE INDEX
<table>
<thead>
<tr>
<th>Token</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>STREAM</td>
<td>3-5 (54)</td>
</tr>
<tr>
<td>stream-io-statement</td>
<td>3-11 (122), 3-7 (78)</td>
</tr>
<tr>
<td>stream-optionslist</td>
<td>3-11 (123), 3-11 (122)</td>
</tr>
<tr>
<td>STRING</td>
<td>3-11 (123)</td>
</tr>
<tr>
<td>string-attribute</td>
<td>3-3 (34), 3-2 (23), 3-3 (31), 3-9 (103)</td>
</tr>
<tr>
<td>string-character</td>
<td>3-14 (162), 3-14 (161)</td>
</tr>
<tr>
<td>string-format</td>
<td>3-6 (69), 3-6 (65)</td>
</tr>
<tr>
<td>STRINGRANGE</td>
<td>3-7 (75)</td>
</tr>
<tr>
<td>STRINGSIZE</td>
<td>3-7 (75)</td>
</tr>
<tr>
<td>SUB</td>
<td>3-13 (152)</td>
</tr>
<tr>
<td>subscriptlist</td>
<td>3-13 (149), 3-12 (143)</td>
</tr>
<tr>
<td>SUBSCRIPTRANGE</td>
<td>3-7 (75)</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>3-2 (20), 3-9 (105)</td>
</tr>
<tr>
<td>T</td>
<td>3-13 (149), 3-14 (166)</td>
</tr>
<tr>
<td>TASK</td>
<td>3-3 (31), 3-8 (91)</td>
</tr>
<tr>
<td>THEN</td>
<td>3-8 (87)</td>
</tr>
<tr>
<td>TITLE</td>
<td>3-10 (119)</td>
</tr>
<tr>
<td>TO</td>
<td>3-8 (85), 3-9 (89)</td>
</tr>
<tr>
<td>TRANSIENT</td>
<td>3-5 (54)</td>
</tr>
<tr>
<td>TRANSMIT</td>
<td>3-10 (112)</td>
</tr>
<tr>
<td>U</td>
<td>3-13 (149)</td>
</tr>
<tr>
<td>UNALIGNED</td>
<td>3-3 (31)</td>
</tr>
<tr>
<td>UNBROTTED</td>
<td>3-5 (54)</td>
</tr>
<tr>
<td>unconditional-statement</td>
<td>3-7 (78), 3-6 (72), 3-8 (88), 3-9 (105)</td>
</tr>
<tr>
<td>UNDEFINEDFILE</td>
<td>3-10 (112)</td>
</tr>
<tr>
<td>UNDERFLOW</td>
<td>3-7 (75)</td>
</tr>
<tr>
<td>UNLOCK</td>
<td>3-11 (130)</td>
</tr>
<tr>
<td>unsubscripted-reference</td>
<td>3-13 (149), 3-3 (29), 3-5 (58), 3-10 (109), 3-10 (110)</td>
</tr>
<tr>
<td>UPDATE</td>
<td>3-5 (54)</td>
</tr>
<tr>
<td>V</td>
<td>3-13 (149), 3-14 (166)</td>
</tr>
<tr>
<td>VALUE</td>
<td>3-2 (21)</td>
</tr>
<tr>
<td>value-clause</td>
<td>3-2 (20)</td>
</tr>
<tr>
<td>value-spec</td>
<td>3-2 (21)</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>3-3 (31)</td>
</tr>
</tbody>
</table>
VARING
VOLUME
W
WAIT
wait-statement
WHEN
WHILE
WRITE
X
Y
Z
ZERODIVIDE
0
1
2
3
4
5
6
7
8
9