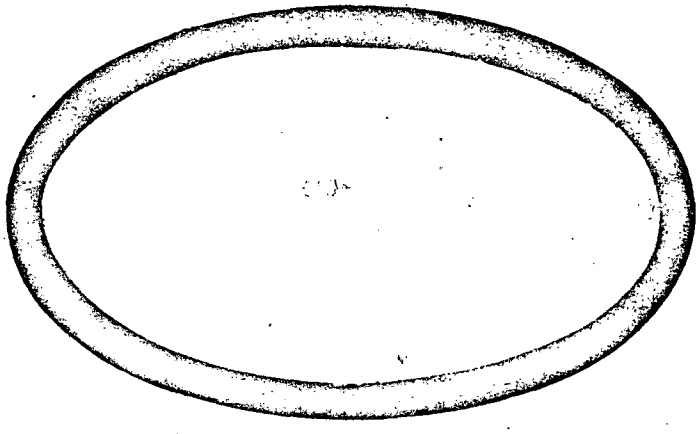
CR115515

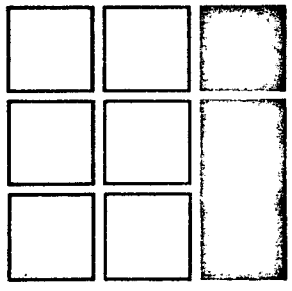


(NASA-CR-115515) ADVANCED SOFTWARE TECHNIQUES FOR DATA MANAGEMENT SYSTEMS. VOLUME 3: PROGRAMMING LANGUAGE T.A. James, et al (Intermetrics, Inc.) Feb. 1972 288 p

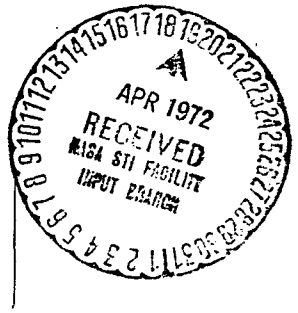
N72-21203

CSCL 09B G3/08

Unclas 23752



REPRODUCED BY
NATIONAL TECHNICAL
INFORMATION SERVICE
U. S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161



INTERMETRICS

OFFICE OF PRIME RESPONSIBILITY

EDS

CAT. 08

FOREWARD

This document is Volume III of a Final Report for contract NAS9-11778, Advanced Software Techniques for Data Management Systems. It contains a summary of the features and characteristics of eight higher order programming languages: PL1, HAL, FORTRAN, JOVIAL, CLASP, SPL, ALGOL, MAC.

The report was prepared as part of a programming language evaluation task within contract NAS9-11778, with a portion of its content collected during contract NAS9-10542 entitled Development of an MSC Language and Computer.

The overall study contract was performed for the Manned Spacecraft Center, Houston, Texas. It was accomplished during the period from 16 June 1971 through 1 February 1972, by Intermetrics, Inc., under the technical direction of Mr. Joseph Saponaro. The technical monitor for the Manned Spacecraft Center was Mr. Donald Barron/EB-5.

The publication of this report does not constitute approval by the NASA of the findings or recommendations contained therein.

Volume III Final Report
Contract NAS9-11778

ADVANCED SOFTWARE TECHNIQUES FOR DATA MANAGEMENT SYSTEMS

February 1972

PROGRAMMING LANGUAGE CHARACTERISTICS AND COMPARISON REFERENCE

Prepared by:

T. A. James
B. C. Hall
P. M. Newbold

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

TABLE OF CONTENTS

		Page
I.	INTRODUCTION	1
II.	OUTLINE OF COMPARISON	3
III.	SUMMARY OF COMPARISON	14
IV.	GLOSSARY	22
V.	REFERENCES	24
VI.	GROUP I LANGUAGE TABLE	
VII.	GROUP II LANGUAGES TABLE	
VIII.	GROUP III LANGUAGES TABLE	

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

I. Introduction

The comparitive evaluation of programming languages has been characterized by inadequate criteria of comparison, and by insufficiently defined methods of demonstrating their differences. A more comprehensive approach to language evaluation, taken by Sammet [1] for example, is to define an organized set of technical characteristics considered to be relevant to the evaluation, and then to describe each language in terms of these characteristics. Examples of such technical characteristics may be the character set used in the language, types of variables, types of statements, and so on.

By this means one may at least arrive at adequate criteria for comparing languages, but there remains the problem of codifying the information gathered so as to be able to arrive at an understanding of its meaning. Obviously, if a number of higher order languages of the customary degree of complexity are being compared, the sheer bulk of information is too great to allow of a narrative style of comparison. The solution is to tabulate the descriptions under the heading of each characteristic in turn. It is then a simple matter to compare one language with another according to the purpose one has in view. The advantages of using the tabular form can be summed up as follows:

- conciseness and clarity in classification.
- completeness by revealing areas of deficiency of languages.
- ease of cross-referencing.

The use of tables, even though bringing a detailed comparison down to manageable proportions, cannot provide entirely adequately an overview of the major features of the languages being evaluated. For this purpose what is required is a summary of the content of the comparison tables, including a rough quantitative "merit rating" of each feature. To illustrate the meaning of the merit rating, consider three languages whose vector-matrix manipulation features are being compared. Language A might have a full range of vector-matrix operations available, including inversion and computation of the adjoint. Language B might have no explicit-vector matrix operations as such, but possess features from which they could be synthesized with some degree of effort. Language C might be completely intractable. On a scale of 0 to 10 one might rate the languages

on this feature as follows:[1]

Language	Score
A	10
B	3
C	0

This document makes a comparative evaluation of eight higher order languages of general interest in the aerospace field:

* PL/1
* HAL
* JOVIAL/J3
* SPL/J6
* CLASP
* ALGOL 60
* FORTRAN IV
* MAC 360

A summary of the functional requirements for a language for general use in manned aerodynamic applications is presented in Figure 1.1. The approach described in the previous paragraphs is taken in their evaluation; the major features in the language are given merit ratings on the scale 0 to 10.

The purpose of the evaluation is to supply background material with the help of which the worth of each language in some particular application can be assessed. A suggested strategy which one might employ in using the evaluation is to weigh each merit rating by an "importance factor" for the given application. The content of this document hopefully reflects no bias in application itself (except of course in the selection of the languages to be compared). The languages that are new and evolving may change considerably due to new additions or extensions. Such changes, of course, will not be reflected in the tables.

The remainder of the document is laid out in the following way. Section II gives a synopsis of the technical characteristics used as a basis of comparison, together with various explanatory notes relating to the comparison tables themselves, which may be found in the appendix.

1. It might be argued that generating ratings in this way is valueless since to a large extent it is intuitive and subjective. In defense, it is asserted that the value of a subjective judgement lies in the experience and authority with which it is made. In practice it has been found that a meaningful consensus of opinion on the merits of a language's features can be arrived at. Those especially sceptical can always consult the tables and come to an independent judgement.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Figure 1.1 Functional Requirements

Software Functions	Description/Comments	Predominant Language/Compiler Characteristics
1. Navigation, Guidance, Targeting, General Mission Programs	1. Orbit determination of position and velocity using sensor measurements. 2. Outer loop steering commands. 3. Maneuver calculation & trajectory control. 4. Position and velocity extrapolation. 5. Abort trajectory calculations. 6. Radar tracking. 7. Boost and entry aerodynamic control.	1. Highly mathematically oriented - extensive use of vector and matrix algebra. 2. Wide dynamic range of numeric data variables (i.e. accuracy, precision needed). 3. Boolean variable and flags for logical decisions (e.g. hardware, mission and system status checks). 4. Real Time Control. • Time critical computations • Multi-programming • Precise event timing • Task synchronization
2. Control and Stabilization	1. Attitude control 2. Accelerated flight 3. Balance control 4. Automatic landing	Same as above with somewhat more emphasis on boolean logical decisions.

Functional Requirements (cont.)

Software Functions	Description/Comments	Predominant Language/Compiler Characteristics
3. Operating Systems	1. Executive control functions: scheduling, resource allocation, dispatching, and queue processing. 2. Supervisory functions: • I/O control • Interrupt processing • Central service routines	1. Simple scalar arithmetic, no vectors or matrices. 2. Boolean variables and logical decision making. 3. Lists and pointers. 4. Arrays and tables of data. 5. Hardware interfaces. 6. Character processing for text, names and messages. 7. Relational operators for all data comparisons. 8. Storage allocation. 9. I/O statements. 10. Interlocking of data storage.
4. On-board Checkout and System Monitoring	1. Status monitoring of subsystem hardware. 2. On- and off-line diagnostics.	1. Simple mathematical operations. 2. Hardware interfacing. 3. Bit manipulation for complex logical decisions.

Functional Requirements (cont.)

Software Functions	Description/Comments	Predominant Language/Compiler Characteristics
4. Onboard Checking and System Monitoring (continued)	3. Fault predictions, e.g. trend analysis. 4. Subsystem testing. 5. Reconfiguration control.	4. Data comparisons, relational operators. 5. Interfaces with machine language code.
5. Data Management	1. General file handling for stored data. • Retrieval • Updating • Reformating • Deleting • General file maintenance 2. Processing of large volumes of measurement data.	1. Hierarchical organizations of data, lists and arrays of data. 2. Mathematical reduction & processing of data. 3. Pointers - file directories 4. Test and character handling. 5. Storage allocation.
6. Communications & Display	1. Displays • Terminals, keyboard and console support programs	1. Bit manipulation. 2. Character strings and operations. 3. Logical operations.

Functional Requirements (cont.)

Software Functions	Description/Comments	Predominant Language/Compiler Characteristics
6. Communications & Display (continued)	2. Communications. • Message processing, formating, etc. • Uplink, downlink • Telemetry processing 3. CRT's and graphics. 4. Special pilot displays	4. Arrays and data tables. 5. Hardware I/O.
7. Compiler & Utility Support Software	1. Compilers, assemblers, and diagnostic packages.	1. Arrays and lists of data. 2. Character and text processing. 3. Boolean variables and operations. 4. Relational operators for comparisons. 5. Input/output control.

NOTE:

The functions above refer to flight software as well as ground support and analysis, including simulation. For example, design and development of guidance techniques require many of the same language characteristics as the final flight software product.

Section III summarizes the content of the tables, and gives merit ratings for the features of the languages.

II. Outline of Comparison

In this section the set of technical characteristics used in the language comparison are explained. For the purposes of the comparison the languages are divided into three groups as follows:

Group I: multi-purpose languages

PL/1
HAL

Group II: aerospace/command and control languages.

JOVIAL/J3
SPL/J6
CLASP

Group III. general scientific languages

ALGOL 60
FORTRAN IV
MAC 360

The appendix contains separate comparison tables for each language group, each prefaced by a brief resume of the languages comprising the group.

The organized set of technical characteristics (six classes) used as a basis for the comparison tables has the following structure:

1. Form of Language- comprising the physical and conceptual properties of the language itself as distinct from those of programs written in the language.

2. Structure of Program- pertaining to the way in which programs are constructed from the elemental parts of the language.

3. Data Element Types, Groups and Operations- describing what data can be handled in the language, and what operations can be performed on it.

4. Executable Statements- categorizing the imperative statements causing or controlling operations on data.

5. Non-Executable Statements- categorizing the declarative statements supplying information about data, storage allocation, external environment and numerous other items.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

6. Structure of Language and Compiler Interaction-relating to characteristics of the language which interact with compiler properties, or with its use.

These six major classes are further subdivided as follows:

1.0 Form of Language

1.1 Character Set

1.1.1 Upper Case Letters

1.1.2 Lower Case Letters

1.1.3 Arabic Numerals

1.1.4 Special Marks or Signs

1.2 Types of Basic Elements

1.2.1 Language Defined Elements

1.2.1.1 Keywords or Primitives

1.2.1.2 Operators

1.2.1.2.1 Physical Graphics

1.2.1.2.2 Keywords

1.2.1.3 Punctuation

1.2.1.3.1 Brackets

1.2.1.3.1.1 Physical Graphics

1.2.1.3.1.2 Keywords

1.2.1.3.2 Separators

1.2.1.3.2.1 Physical Graphics

1.2.1.3.2.2 Keywords

1.2.2 Programmer Defined Elements

1.2.2.1 Identifiers

1.2.2.2 Constants (Literals)

1.2.2.2.1 Numeric

1.2.2.2.2 Textual

1.2.2.2.3 Boolean

1.2.2.2.4 Other

1.2.2.3 Comments

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

1.3 Identifier Definition

1.3.1 Types of Identifiers

1.3.1.1 Statement

1.3.1.2 Data Names

1.3.1.3 Index Variable Names

1.3.1.4 Label Variable Names

1.3.1.5 Others

1.3.2 Formation Rules for Identifiers

1.3.3 Use of Reserved Words

1.3.4 Synonyms

1.3.4.1 Preset

1.3.4.2 Dynamic

1.3.5 Structure of Data Names

1.3.5.1 Subscription

1.3.5.1.1 Dimensions

1.3.5.1.2 Classes

1.3.5.1.3 Form

1.3.5.1.4 Allowable Data Elements

1.3.5.2 Qualification

1.3.5.3 Qualification and Subscription

1.3.5.4 Numbering Conventions

1.3.6 Scope of Names Based on or Relative to Program Structure

1.3.6.1 System Names (Shared)

1.3.6.2 Global Names (Shared)

1.3.6.3 Local Names

1.4 Definition and Usage of Other Basic Elements

1.4.1 Operators

1.4.1.1 Arithmetic Operators

1.4.1.1.1 Scalar

1.4.1.1.2 Non-scalar

1.4.1.2 Comparison Operators

1.4.1.2.1 Binary

1.4.1.2.2 Unary

1.4.1.3 Boolean Operators

1.4.1.3.1 Binary
1.4.1.3.2 Unary
1.4.1.4 Bit String Operators
1.4.1.4.1 Binary
1.4.1.4.2 Unary
1.4.1.5 Functional Modifiers
1.4.1.6 Other
1.4.2 Delimiters
1.4.2.1 Brackets
1.4.2.2 Separators
1.4.3 Punctuation
1.4.4 Significance of Blanks
1.4.5 Literals
1.4.6 Noise Words

1.5 Input Format
1.5.1 Physical Input Format
1.5.1.1 Linear
1.5.1.1.1 Fixed (Columnar Restrictions)
1.5.1.1.2 String
1.5.1.1.3 Combination
1.5.1.2 Non-Linear
1.5.1.2.1 Fixed (Columnar Restrictions)
1.5.1.2.2 String
1.5.1.2.3 Combination
1.5.2 Conceptual Form
1.5.2.1 Symbolic or Formal
1.5.2.2 English-Like
1.5.2.3 Pseudo English-Like

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.0 Structure of Program

2.1 Statement Types

- 2.1.1 Non-Executable Statements
  - 2.1.1.1 Declaratives
  - 2.1.1.2 Compiler Directives
  - 2.1.1.3 Comments
- 2.1.2 Executable Statements
  - 2.1.2.1 Smallest Executable Statement
  - 2.1.2.2 Grouped Executable Statement
    - 2.1.2.2.1 Block Structure
    - 2.1.2.2.2 Others
  - 2.1.2.3 Loops
  - 2.1.2.4 Procedures, Functions or Subroutines
  - 2.1.2.5 Inclusion of Other Languages
    - 2.1.2.5.1 Assembly Language (AL)
    - 2.1.2.5.2 Assembly Routines
    - 2.1.2.5.3 Higher Order Language (HOL)
    - 2.1.2.5.4 HOL-AL Communication
- 2.1.3 Intermingling Order for Non-Executable and Executable Statements
- 2.1.4 Operating System Interface

2.2 Statement Characteristics

- 2.2.1 Methods of Delimiting
  - 2.2.1.1 Explicit and/or Contextual
  - 2.2.1.2 Implicit
- 2.2.2 Parameter Passage Required (Different Data Types)
  - 2.2.2.1 Call by Name
    - 2.2.2.1.1 Formal
      - 2.2.2.1.1.1 Input
      - 2.2.2.1.1.2 Output
    - 2.2.2.1.2 Calling
      - 2.2.2.1.2.1 Input
      - 2.2.2.1.2.2 Output

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.2.2.2 Call by Value
2.2.2.2.1 Formal
2.2.2.2.1.1 Input
2.2.2.2.1.2 Output
2.2.2.2.2 Calling
2.2.2.2.2.1 Input
2.2.2.2.2.2 Output
2.2.2.3 Call by Address
2.2.2.3.1 Formal
2.2.2.3.1.1 Input
2.2.2.3.1.2 Output
2.2.2.3.2 Calling
2.2.2.3.2.1 Input
2.2.2.3.2.2 Output
2.2.3 Embedding
2.2.4 Recursion
2.2.5 Reentrant
2.2.5.1 Serially Reusable
2.2.5.2 Reenterable
2.2.6 Pure Procedure

3.0 Data Element Types, Groups and Operations

3.1 Types of Data Elements
3.1.1 Scalar
3.1.1.1 Arithmetic
3.1.1.1.1 Integer
3.1.1.1.2 Fixed Point (Mixed Number)
3.1.1.1.3 Floating Point
3.1.1.1.4 Binary
3.1.1.1.5 Octal
3.1.1.1.6 Decimal
3.1.1.1.7 Hexadecimal
3.1.1.1.8 Multiprecision
3.1.1.1.9 Other

3.1.1.2 Boolean
3.1.1.3 List or Pointer
3.1.1.4 Other
3.1.2 Non-Scalar
3.1.2.1 Arithmetic
3.1.2.1.1 Vector
3.1.2.1.2 Matrix
3.1.2.1.3 Complex
3.1.2.1.4 Other
3.1.2.2 Bit String
3.1.2.3 Text String

3.2 Groups of Data Elements

3.2.1 Arrays

3.2.2 Hierarchical Structures

3.2.3 Combinations of Above

3.2.4 Files

3.3 Operations With Data Element Types and Groups

3.3.1 Intermingling Rules for Mixed Data Types
3.3.2 Conversion Rules
3.3.3 Alignment Rules
3.3.4 Precision and Computation Rules
3.3.5 Precedence and Sequencing Rules

3.4 Accessibility of Data

3.4.1 Hardware Defined
3.4.1.1 Bits
3.4.1.2 Bytes
3.4.1.3 Words
3.4.1.4 Others
3.4.2 Language Defined
3.4.2.1 Variables and Constants
3.4.2.2 Arrays
3.4.2.3 Hierarchical Structures
3.4.2.4 Combinations

3.5 Scope of Data

4.0 Executable Statements

4.1 Assignment, Exchange and Computation Statements

4.1.1 Data Element Types

4.1.1.1 Scalar

4.1.1.1.1 Arithmetic

4.1.1.1.2 Boolean

4.1.1.1.3 Comparison

4.1.1.1.4 List or Pointer

4.1.1.1.5 Other

4.1.1.2 Non-Scalar

4.1.1.2.1 Arithmetic

4.1.1.2.2 Bit String

4.1.1.2.3 Text String

4.1.2 Arrays

4.1.3 Hierarchical Structures

4.1.4 Nested Assignment (Factoring)

4.1.5 Conversion Rules for Results

4.2 Textual Data Handling Statements

4.2.1 Editing

4.2.2 Conversion

4.2.3 Sorting

4.2.4 Comparison

4.3 Sequence Control and Decision Making Statements

4.3.1 Unconditional Control Transfer

4.3.1.1 No Return

4.3.1.2 Return

4.3.2 Conditional Control Transfer

4.3.2.1 No Return

4.3.2.2 Return

4.3.3 Loop or Index Control

4.3.3.1 Loop Designation

4.3.3.2 Range of Loops

4.3.3.3 Iteration Control Mechanism

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

4.3.4 Real Time Control
4.3.4.1 Multitasking
4.3.4.2 Scheduling and Dispatching
4.3.4.3 Storage Allocation
4.3.4.3.1 Static
4.3.4.3.2 Dynamic
4.3.4.4 Access Restrictions
4.3.4.4.1 Memory
4.3.4.4.2 Registers
4.3.4.4.3 Interrupts
4.3.4.5 Interrupt Handling
4.3.5 Error Condition and Program Checking

4.4 Symbolic Data Handling Statements
4.4.1 Algebraic Expression Manipulation
4.4.2 List Handling
4.4.3 String Handling
4.4.4 Pattern Handling

4.5 Interaction With Operating System and/or Environment Statements
4.5.1 Input/Output
4.5.1.1 File Initialization/Processing/Termination
4.5.1.2 File Positioning and Handling
4.5.2 Library Reference
4.5.3 Debugging
4.5.4 Storage and Segmentation Allocation
4.5.4.1 Static
4.5.4.2 Dynamic
4.5.5 Operating System and Machine Dependent
4.5.6 Others

5.0 Non-Executable Statements

5.1 Data Declarations
5.1.1 Data Element Type Declarations

5.1.1.1 Constants

5.1.1.1.1 Arithmetic
5.1.1.1.1.1 Hexadecimal
5.1.1.1.1.2 Decimal
5.1.1.1.1.3 Octal
5.1.1.1.1.4 Binary
5.1.1.1.2 Boolean
5.1.1.1.3 Textual
5.1.1.1.4 Other
5.1.1.2 Variables
5.1.1.2.1 Arithmetic
5.1.1.2.2 Boolean
5.1.1.2.3 Textual
5.1.1.2.4 Other
5.1.1.3 Presetting of Declarations
5.1.1.4 Nesting (Factoring) of Declarations
5.1.1.5 Default Options
5.1.1.6 Numbering Conventions
5.1.2 Group Type Declarations
5.1.2.1 Array Declarations
5.1.2.2 Hierarchical Structures
5.1.2.3 Procedure, Function, Subroutine Declarations
5.1.2.4 Presetting
5.1.2.5 Nesting (Factoring) of Declarations
5.1.2.6 Default Options
5.1.2.7 Numbering Conventions
5.1.3 Interaction With Operating System and/or Environment
5.1.3.1 File Declarations
5.1.3.2 Storage and Segmentation Declarations
5.1.3.3 Hardware Declarations
5.1.3.4 Format Declarations

5.2 Compiler Directives

5.2.1 Optimization

5.2.1.1 Space

5.2.1.2 Time

5.2.2 Debugging Aids

5.2.3 Documentation

5.2.4 Documentation Control

5.2.5 Storage Control

5.2.6 Compilation Control

5.2.7 Others

6.0 Structure of Language and Compiler Interaction

6.1 Self Modification of Programs

6.2 Bootstrapping

6.3 Extensible

6.4 Subsets or Dialects

6.5 Debugging, Parametric Programming Facilities

6.5.1 Compilation Time

6.5.2 Object Time

6.6 Effect of Language Design on Implementation Efficiency, Remarks

Some of the headings found in the list may be unfamiliar: a glossary of the rarer terms may be found in Section IV. To keep the comparison tables down to a manageable size, the content is presented in a much abbreviated form. In cases of difficulty the relevant user guide listed in the table of references should be consulted. If a technical characteristic does not exist in a particular entry, 'NO' is entered in the table.

III. Summary of Comparison

In this section the content of the comparison tables set out in Secs. VI-VIII are summarized. In order to make the summary more meaningful, the technical characteristics of the eight languages compared have been regrouped into 17 major language features. For each of the features, a few brief comparative remarks are made, and the languages' merit ratings given. All eight languages are compared without division into the three groups specified in Section II.

It is stressed at this point that the merit ratings given reflect the degree to which the languages possess comprehensive coverage of the language features, but do not reflect the ease of use of the language features. To give an example, consider three languages A, B, and C. Suppose that A and B have a full vector-matrix arithmetic facility. Suppose also that the languages can be ranked C, B, A in terms of ease of use of the facility. Then A and B would both be given an equal, high merit rating, but C a low rating.

It is also important to bear in mind when interpreting the merit ratings, that the figures given are not entirely relative: the language giving best coverage of a feature is not automatically given a rating of 10, or the worst a rating of 0. However in many instances the best language does get a high merit rating of 8-10, because it has in these cases been judged to be very adequate in its coverage of the feature in question. Very low ratings may indicate that the language is outside its recognized field of application.

The merit ratings for each feature are given in the following form:

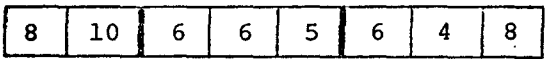
P	H	J	S	C	A	F	M

In place of each letter the merit rating (on the scale 0-10) for the corresponding language is given. The letters have the following connotation:

P = PL/1	C = CLASP
H = HAL	A = ALGOL 60
J = JOVIAL/J3	F = FORTRAN IV
S = SPL/J6	M = MAC-360

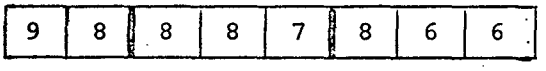
The broad black divisions indicate divisions between the three language groups.

1. Basic Language Form



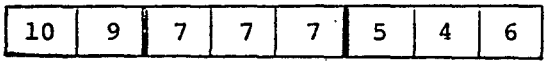
All languages except A,F take advantage of a full standard character set. F uses a very restricted character set; A a restricted set with some unusual symbols. All languages except F are stream oriented, but P,H have the fewest restrictions. H,M have multiline inputs which makes source input more like the problem statement and thus easier to use. Languages contain certain numbers of keywords related to their complexities; in H,J,C,M these are reserved words. C,F,M have the most restrictions on the formation of identifiers. F contains the least punctuation.

2. Program Structure



All languages except S,C,F have a hierarchical block structure. Only S,C have provision for inline machine code. P,H,A,M allow integrally compiled procedures and functions, possibly nested; in J,S,C such procedures and functions exist but cannot be nested; in F procedures and functions are generally separately compiled. In addition H allows separately compiled programs to be invoked, and M separately compiled procedures. P,S,A have recursion, P,S, at the user's option. S allows reentrancy at the user's option. P,H,C have call-by-address and call-by-value; J,S,A have call-by-name and call-by-value (A at user option); F has call-by-address, and M has its own version of call-by-value.

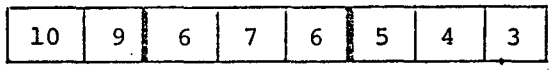
3. Data Elements



All languages have integer and scalar data elements except M, which has "index" data elements instead of integer, usable only in a restricted way. All languages except M have a form of Boolean data element. H,J,S,C,M have vector and matrix elements. P,F have complex data elements. P,H,J,S,C have fixed-point as well as floating-point elements. In addition J,S,C have location data elements.

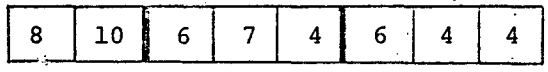
P,H have bit strings and character strings; J,S,C,A have character strings only, A allowing very limited use. A,F,M give no access to hardware levels of data, other languages a varying degree. P,H,S,C,F allow precision of certain data elements to be specified explicitly.

4. Data Aggregates



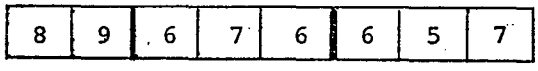
All languages possess the equivalent of multidimensional arrays of data. In addition, P,H,S,C permit hierarchical (tree-like) organizations of data elements. P,H,S,C also allow combinations of arrays and hierarchical organizations, C however, only in a limited way.

5. Subscripting of Elements and Aggregates



All languages allow subscripting, C,F,M only of very limited form. P,H,J,S allow subscripted subscripts. P,H,S,C subscripting includes a cross-section facility. Only H subscripting includes full partitioning ability on subscripts of any level, for elements (where applicable) and aggregates.

6. Arithmetic Processing



F does not allow mixing of integer and scalar types: other languages have no special restrictions on authentic statements. P,H,S,C,A allow assignments to multiple receivers, sometimes of mixed type. P,A allow conditional sub-expressions to be intermixed. J,S,C have symmetrical exchange statements. Of the languages which permit fixed-point operation, P,J,S,C allow mixing with floating point. Only P,A,F possess integer division capability. P,H have comprehensive facility for arithmetic processing on array aggregates; S,C,M a very limited facility.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

7. Processing of Text and Bit Strings

9	9	5	5	5	1	2	0

A,F,M have neither bit string nor text string processing abilities. J,S,C do not have a bit string processing ability. J,S,C can only do string assignments and conversions. P,H have a catenation and substring selection facility. Only H has a text string comparison facility. None of the languages can directly sort or edit text strings. Of those languages which handle strings, only P,H have a comprehensive ability to operate on array aggregates of strings.

8. Sequence Control and Decision Making

10	9	7	7	7	7	4	7

All languages have the equivalent of an unconditional 'goto' and 'do case' ('computed goto'). All languages except F have an 'if-then-else' structure or its equivalent. F has only the equivalent of the 'if-then' structure. All languages have the capacity to specify nested 'do-groups' in some form. J,S,C, F,M can include only one range specification for each do-group. P,H,A can include a list of range specifications to be executed serially. In P,A a 'while' clause can optionally be appended to each item in the list: in H at the end of the list only. Only F does not allow negative increments. P,H,J,S,C,A have also a 'do-while' facility or its equivalent.

9. Real Time Processing

9	8	1	5	5	0	0	0

J,A,F,M have no real time processing capability. P,H have a multi-tasking facility; and S a limited facility of a similar type. P,H have a scheduling and dispatching facility; S a limited facility. P,H,S,C, all have locking, interrupt and error-handling capabilities of some degree of sophistication.

10. List and Symbolic Data Handling

6	1	0	0	0	0	0	0

J,S,C,A,F,M have no facilities whatsoever. P,H have no symbolic data handling facility as such, but elementary list processing operations can be synthesized in P with some difficulty.

11. Input-Output Operations

10	6	6	6	1	1	7	5

Comparison of the I/O facilities provided by different languages is very difficult, because languages are very diverse in these features. Only very broad comments will be made here: the above merit ratings should be viewed with a healthy scepticism. C,A provide no I/O features as an inherent part of the language. M provides the limited purpose-oriented I/O features READ, PRINT and PUNCH. J,S provide record-oriented device independent file I/O with provision for specifying the file parameters. F provides device-independent I/O oriented towards non-direct access devices (extensions exist). P,H provide device-independent I/O for direct-access and other devices. In P file specifications can be made. In P,H,J,S,F the logical device can be specified. On input, H is stream oriented; F is format oriented and P,M have both capabilities. Similar remarks are true of output.

12. Data Declarations

10	10	7	8	7	6	6	5

All languages provide data declaration statements commensurate with their needs. P,H,F,M have default (implicit) declaration capabilities. P,H,S,C,F allow factored declarations. P,H,J,S,C allow data to be intialized at the time of declaration. F has an equivalent facility with a separate statement. Data elements can be made constant (read only) in P,H,J,S,C.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

13. Formatting of I/O

9	6	0	1	0	0	9	7

J,S,C,A have no formatting capability. P,F provides separate format statements to control position and form of each item input or output. H provides control over position of data items only. M has a limited formatting capability.

14. Compiler Optimization Directives

3	0	0	9	9	0	0	0

Consideration is not given to compilers for a language having an optimization feature, but only whether the control of such a feature can be specified in the language. In this sense only P,S,C have optimization directives, P only indirectly.

15. Debugging Aids

6	1	0	9	9	0	0	8

J,A,F have no debugging aids built into the language (some implementations of F have been extended to include a trace facility). S,C,M have trace facilities. Programs written in P,H can include actions to be taken on occurrence of certain errors: P provides an optional trace of subprogram calls at such time.

16. Extensibility of Language

5	3	1	2	0	0	0	0

C,A,F,M have no extensibility properties. H,J,S have replacement operations by which a name may represent statements or groups thereof, giving a rudimentary ability. P has a compile-time macro facility which gives a little more power. No free designed-in extensibility exists however.

17. Functions Built Into Language

9	8	4	5	3	4	7	5

J,C,A are poor in built-in functions; S is better, P,H,F,M are relatively prolific. Ratings given are based on supposition that only functions which can be used in a language are considered significant (for example F is not penalized for not having a LENGTH function because F has no string data).

This concludes the summary of the language comparison set out in full in the tables in Section VI - VIII. Note again that the merit ratings given above reflect the breadth of the languages in each area of consideration, and not necessarily the utility of that area. Figure III.1 sets out again the merit ratings of the languages.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

	P	H	J	S	C	A	F	M	
1. Basic Language Form	8	10	6	6	5	6	4	8	1
2. Program Structure	9	8	8	8	7	8	6	6	2
3. Data Elements	10	9	7	7	7	5	4	6	3
4. Data Aggregates	10	9	6	7	6	5	4	3	4
5. Subscripting of Elements (Aggregates)	8	10	6	7	4	6	4	4	5
6. Arithmetic Processing	8	9	6	7	6	6	5	7	6
7. Processing of Text and Bit Strings	9	9	5	5	5	1	2	0	7
8. Sequence Control and Decision Making	10	9	7	7	7	7	4	7	8
9. Real Time Processing	9	8	1	5	5	0	0	0	9
10. List and Symbolic Data Handling	6	1	0	0	0	0	0	0	10
11. Input-Output Operations	10	6	6	6	1	1	7	5	11
12. Data Declarations	10	10	7	8	7	6	6	5	12
13. Formatting of I/O	9	6	0	1	0	0	9	7	13
14. Compiler Optimization Directives	3	0	0	9	9	0	0	0	14
15. Debugging Aids	6	1	0	9	9	0	0	3	15
16. Extensibility of Language	5	3	1	2	0	0	0	0	16
17. Functions Built into Language	9	8	4	5	3	4	7	5	17

FIGURE III.1 MERIT RATINGS

Glossary

BOOTSTRAPPING: the process of writing a compiler for a language in the language to be compiled.

CALL BY ADDRESS: passage of an address into a procedure or function to substitute for the formal parameter at the time of execution.

CALL BY NAME: passage of the name of a variable into a procedure or function to substitue for the formal parameter at execution time.

CALL BY VALUE: passage of the value of a variable or expression into a procedure or function to substitute for the formal parameter.

COPROCEDURES: two procedures P and Q are coprocedures if P can call Q and return, and also Q can call P and return.

DELIMITERS: parts of a language serving the syntactic purpose of aiding the definition or identification of other language elements.

DOCUMENTATION: in the sense of the comparison tables, the capacity of a language to issue compiler directives to print symbol tables, cross-reference listings and so on.

DOCUMENTATION CONTROL: the capacity of a language to control the format of the printed matter constituting documentation (in the sense of the glossary).

EMBEDDING: the use of statements or statement groups as part of a larger statement.

EXTENSIBLE: a language is extensible if in one program unit one can specify entities which can then be used as if they were new permanent language elements (authentic statement functions in FORTRAN, for example).

FACTORING: the process of grouping a number of data items together to be operated on, or assigned attributes in an identical way.

FUNCTIONAL MODIFIERS: a class of inline built-in functions possessed in the JOVIAL language.

LINEAR INPUT: the use of only one line at a time to write source program statements, not more than one simultaneously.

NOISE WORD: a keyword having no syntactic use, put in to clarify a meaning to the user of the language.

NON-LINEAR INPUT: the simultaneous use of more than one line to write source program statements (for example the use of subscript lines).

PARAMETRIC PROGRAMMING: the use of symbolic statements which generate source language statements at compile time.

PHYSICAL GRAPHICS: Special characters other than the alphanumerics, used as delimiters, operators and so on.

PURE PROCEDURES: procedures which do not modify themselves during execution.

RECURSION: the ability of a procedure to call itself, or more generally of a chain of successive calls to be closed (X calls Y calls Z calls X).

REENTERABLE: the ability of a procedure to be used concurrently by more than one caller.

REENTRANT: the ability of a procedure to handle calls and returns when being used concurrently by more than one caller.

SERIALLY REUSABLE: the ability of a procedure to look as if it had not be modified during use when the return is made.

REFERENCES

1. Sammet, J.E., Programming Languages: History and Fundamentals, Prentice-Hall, Englewood Cliffs, N.J., 1969.

2. Shaw, C.J., A specification of JOVIAL, Comm. ACM, Vol. 6, No. 12 (December 1963), pp. 721-736.

3. SDC, Space Programming Language (SPL/J6) Programmers Manual, SAMSO-TR-68-383, November 1968.

4. Logicon, Incorporated, Flight Computer and Language Processor Study, NAS 12-2005, July 1969.

5. Andersen, C., An Introduction to ALGOL 60, Addison Wesley, Reading, Ma. 1964.

6. Naur, P. (ed.), Revised Report on the Algorithmic Language ALGOL 60, Comm. ACM Vol. 6, No. 1 (Jan., 1963), pp. 1-17.

7. American Standards Association, American Standard FORTRAN, X39-1966, March 7, 1966.

8. MIT/Instrumentation Laboratory, Users Guide to MAC-360, NAS 9-4065, September 1967.

9. IBM, IBM System/360 PL/1 Reference Manual, Form C28-8201-2, October 1969.

GROUP I LANGUAGES

INTRODUCTION

The development of PL/1 results from a desire in the SHARE FORTRAN project for a more complete, less restrictive programming language. The original specification was published in March 1964 and has undergone many revisions since then. The language now resembles a conglomerate of FORTRAN IV, ALGOL 60, and COBOL. Thus PL/1 provides, in a single unified programming language, facilities for scientific/numeric, business and system programming. These areas of application have previously been served by separate languages.

Compilers were started in the spring of 1965 but they only translated subsets of the full language. By late 1967 compilers were beginning to be developed that more closely resembled the full language. Since the language is so large, it is difficult to learn quickly. Therefore most users use only those parts of the language that are necessary for their problems. At this time the use of PL/1 is increasing but it has not replaced FORTRAN IV or COBOL.

HAL language development began in May of 1970 and resulted from a desire in NASA MSC for a language more suitable for solving problems of manned space flight. A compiler for a subset of the language was finished in June 1971. The compiler generated code in the FORTRAN IV language. A code generator is now being developed for the IBM 4π AP computer.

	Form of Language	
1.0	PL/1	HAL
1.1 Character Set		
1.1.1 Upper Case Letters	26 letters of English alphabet - A to Z	A to Z
1.1.2 Lower Case Letters	No	a to z (optional, may be used in identifiers)
1.1.3 Arabic Numerals	10 Arabic numerals - 0 to 9	0 to 9
1.1.4 Special Marks or Signs	60 Character Set	
	$ (currency sign) } defined as	$
	@ (commercial at sign) } alphabetic	@
	# (number sign) }	#
	= (equal sign)	=
	+ (plus sign)	+
	- (minus sign)	-
	* (asterisk)	*
	/ (slash)	/
	((left parenthesis)	(
	) (right parenthesis)	)
	, (comma)	,
	. (period or decimal point)	.
	' (single prime)	'
	% (percent)	;
	; (semicolon)	

1.0	Form of Language	
	PL/1	HAL
	: (colon)	:
	¬ (not)	¬
	& (ampersand)	&
	| (or)	|
	> (greater than)	>
	< (less than)	<
	_ (underscore)	_ (break character)
	? (question mark)	[ ] (brackets)
	b (blank or space)	{ } (braces)
	48 Character Set	4 Characters restricted to character strings
	$ (dollar sign) - defined as alphabetic	
	+ (plus sign)	! (exclamation mark)
	- (minus sign)	% (percent)
	* (asterisk)	? (question mark)
	/ (slash)	" (double quotation mark)
	((left parenthesis)	
	) (right parenthesis)	
	, (comma)	
	. (period or decimal point)	
	' (single prime)	
	= (equal sign)	
	b (blank or space)	

1.0	Form of Language	
	PL/1	HAL
1.2 Types of Basic Elements		
1.2.1 Language Defined		
1.2.1.1 Keywords or Primitives	ABS %ACTIVATE (%ACT) ADD ADDR ALIGNED ALL ALLOCATE ALLOCATION ANY AREA ATAN ATAND ATANH AUTOMATIC (AUTO) BACKWARDS BASED BEGIN BINARY (BIN) BIT BCOL BUFFERED (BUF) BUFFERS BUILTIN BY BY NAME CALL CEIL CHAR CHARACTER (CHAR) CHECK CLOSE COBOL	ACCESS AND ARRAY ASSIGN AT AUTOMATIC BIN BIT BITLENGTH BY CALL CASE CAT CHAR CHARACTER CHARLENGTH CLOSE COLUMN CONSTANT DEC DECLARE DO ELSE END ERROR EVENT EXCLUSIVE FALSE FILE FOR FUNCTION GO

1.0	Form of Language	
	PL/1	HAL
	COLUMN (COL)	HEX
	COMPLETION	IDCODE
	COMPLEX (CPLX)	IF
	COMDITION	IN
	CONJG	INCLUDE
	CONSECUTIVE	INDEPENDENT
	CONTROLLED (CTL)	INITIAL
	CONVERSION (CONV)	INTEGER
	COPY	LABEL
	COS	LATCHED
	COSD	LINE
	COSE	MATRIX
	COUNT	MATRIXDIM
	CTLASA	NOT
	CTL360	NONQUALIFIED
	DATA	OCT
	DATAFIELD	OFF
	DATE	ON
	%DEACTIVE (%DEACT)	OR
	DECIMAL (DEC)	OUTER
	DECLARE (DCL)	PAGE
	%DECLARE (%DCL)	PRECISION
	DEFINED (DEF)	PRIO
	DELAY	PRIOCHANGE
	DELETE	PRIORITY
	DIM	PROCEDURE
	DIRECT	PROGRAM
	DISPLAY	QUALIFIED
	DIVIDE	READ
	DO	READALL
	%DO	REPLACE
	EDIT	RETURN
	ELSE	SCALAR
	%ELSE	SCHEDULE
	EMPTY	SEND
	END	SIGNAL
	%END	SKIP

1.0	Form of Language	
	PL/1	HAL
	ENDFILE ENDPAGE ENTRY ENVIRONMENT (ENV) ERF ERFC ERROR EVENT EXCLUSIVE (EXCL) EXIT EXP EXTERNAL (EXT) F FILE FINISH FIXED FIXEDOVERFLOW (FOFL) FLOAT FLCOR FORMAT FREE FROM G GENERIC GENKEY GET GO TO (GOTO) %GO TO (%GOTO) HBOUND HIGH IF %IF IGNORE IMAG IN %INCLUDE	STATIC SYSTEM TAB TASK THEN TERMINATE TO TRUE UNTIL UPDATE VARYING VECTOR VECTORLENGTH WAIT WHILE WRITE Including Built-in Functions: ABS ABVAL ADJ ARCCOS ARCCOSH ARCSIN ARCSINH ARCTAN ARCTANH BIT CEILING CHARACTER COS COSH DATE DET EXP FLOOR INDEX

1.0	Form of Language	
	PL/1	HAL
	INDEX	INTEGER
	INDEXAREA	INVERSE
	INDEXED	LENGTH
	INITIAL (INIT)	LJUST
	INPUT	LOG
	INTERNAL (INT)	MATRIX
	INTO	MAX
	KEY	MIN
	KEYED	MOD
	KEYFROM	PROD
	KEYTO	RANDOM
	LABEL	RANDOMG
	LENGTH	RJUST
	LBOUND	ROUND
	LEAVE	SCALAR
	LIKE	SIGN
	LINE	SIGNUM
	LINENO	SIN
	LINESIZE	SINH
	LIST	SQRT
	LOCATE	SUBBIT
	LOG	SUBMIT
	LOG2	SUM
	LOG10	TAN
	LOW	TANH
	MAIN	TIME
	MAX	TRACE
	MIN	TRANSPOSE
	MOD	TRUNCATE
	MULTIPLY	UNIT
	NAME	VECTOR
	NCP	
	NOCHECK	
	NOCONVERSION (NOCONV)	
	NOFIXEDOVERFLOW (NOFOFL)	
	NOLOCK	
	NOOVERFLOW (NOOFL)	

1.0	Form of Language	
	PL/1	HAL
	NOSIZE NOSTRINGRANGE (NOSTRG) NOSUBSCRIPTRANGE (NOSUBRG) NOUNDERFLOW (NOUFL) NOWRITE NOZERODIVIDE (NOZDIV) NULL NULLO OFFSET ON ONCHAR ONCOUNT ONCODE ONFILE ONKEY ONLOC ONSOURCE OPEN OPTIONS ORDER OUTPUT OVERFLOW (OFL) PAGE PAGESIZE PENDING PICTURE (PIC) POINTER (PTR) POLY POSITION (POS) PRECISION (PREC) PRINT PRIORITY PROCEDURE (PROC) %PROCEDURE (%PROC) PROD PUT R READ	

1.0	Form of Language	
	PL/1	HAL
	REAL	
	RECORD	
	RECURSIVE	
	REENTRANT	
	REFER	
	REGIONAL	
	RECORD	
	REPEAT	
	REPLY	
	RETURN	
	RETURNS	
	REVERT	
	REWIND	
	REWRITE	
	ROUND	
	SEQUENTIAL (SEQL)	
	SET	
	SIGN	
	SIGNAL	
	SIN	
	SIND	
	SINH	
	SIZE	
	SKIP	
	SNAP	
	SQRT	
	STATIC	
	STOP	
	STREAM	
	STRING	
	STRINGRANGE (STRG)	
	SUB	
	SUBSCRIPTRANGE (SUBRG)	
	SUBSTR	
	SUM	
	SYSIN	
	SYSPRINT	
	SYSTEM	

1.0	Form of Language	
	PL/1	HAL
		individual datum level at the same time. Bit and character string variables may be partitioned into shorter strings or indexed to individual letters or bits. II. Subscripting in combination with built-in shaping functions can be used to control the creation of integers, scalars, bit strings, character strings, vectors, matrices, and arrays of these types from a list of desired components. Subscripting determines the dimensions of vectors, matrices, and arrays and the length of strings. III. Special subscripts can specify precision, or conversion among binary, octal and hexadecimal bit strings.

1.0	Form of Language	
	PL/1	HAL
1.2.1.2 Operators		
1.2.1.2.1 Physical Graphics	60 Character Set	
	+	+
	-	-
	*	*
	/	/
	**	**
	<	<
	<=	<=
	=	=
	>	>
	>=	>=
	¬=	¬=
	¬<	¬<
	¬>	¬>
	&	&
	|	|
	¬	¬
	+	||
	||	.
		#
		@
		$

1.0	Form of Language	
	PL/1	HAL
1.2.1.2.2 Keywords	48 Character Set	
	LT LE GT GE NE NL NG AND OR NOT PT CAT	AND OR NOT CAT TO AT
1.2.1.3 Punctuation		
1.2.1.3.1 Brackets		
1.2.1.3.1.1 Physical Graphics	( ) /* */ ' '	( ) /* */ ' '
1.2.1.3.1.2 Keywords	BEGIN END DO END	DO END

1.0	Form of Language: PL/1	Form of Language: HAL
1.2.1.3.2 Separators		
1.2.1.3.2.1 Physical Graphics	60 Character Set : , = ; b . % ¬	: , = ; . ' b
1.2.1.3.2.2 Keywords	48 Character Set .. ,. BY TO	BY...TO TO

1.0	Form of Language: PL/1	HAL
1.2.2 Programmer Defined		
1.2.2.1 Identifiers	Yes	Yes
1.2.2.2 Constants (Literals)		
1.2.2.2.1 Numeric	Fixed Point Floating Point Integer (Fixed with Zero Fractional Digits) Decimal Binary Complex	Fixed point Floating point with Dceimal, Binary, or Hexadecimal exponentiation Integer
1.2.2.2.2 Textual	Yes	Yes
1.2.2.2.3 Boolean	Yes	Yes
1.2.2.2.4 Other	Pointer, Bit Strings	Binary, octal, hexadecimal, and decimal strings
1.2.2.3 Comments	Comments can be embedded in any statement, where blanks are acceptable except within a literal	Comments can be imbedded anywhere where blanks are allowed, except in character strings, or they can go on separate lines.

1.0	Form of Language: PL/1	Form of Language: HAL
1.3 Identifier Definition		
1.3.1 Types of Identifiers		
1.3.1.1 Statement Names	Yes	Yes
1.3.1.2 Data Names	Yes	Yes
1.3.1.3 Index Variable Names	Yes	Yes
1.3.1.4 Label Variable Names	Yes	No
1.3.1.5 Others	Procedure, Pointer, Bit String, Event, Task, Area names.	Procedure, Function, Program, Task, and Compool names
1.3.2 Formation Rules for Identifiers	All identifiers must begin with an alphabetic character and can be followed by a character string containing no more than 30 alphabetics (26 letters + 3 special + break character) and/or numerics and must be preceded by delimiter and terminated by the (:) delimiter. If identifier referenced no (:) is used.	Identifiers must begin with a alphabetic character followed by alphabetic, numeric and break characters (_), but not ending in a break character. Maximum length is 32. Qualified structure names will have imbedded periods.

1.0	Form of Language: PL/1	HAL
1.3.3 Use of Reserved Words	There are no reserved words that cannot be used as identifiers in the 60 character set. In the 48 character set the graphics (alphabetical equivalents) are considered reserved.	Keywords including names of built-in functions may not be used as identifiers.
1.3.4 Synonyms		
1.3.4.1 Preset	Two or more statement labels can be assigned to a statement and used interchangeably. Through the use of the DEFINED attribute, named variables or constants can be assigned to occupy the same storage area already assigned to another named data item.	This can be accomplished by means of the REPLACE statement which tells the compiler to make a substitution (see 2.1.1.2)
1.3.4.2 Dynamic	No	No
1.3.5 Structure of Data Names		
1.3.5.1 Subscription	Data identifiers (DECLARE) and statement labels can be subscripted. Subscripts are enclosed in ( ) pair and separated with commas. Notation is extended to include concept of cross section (* is used to reference all elements of that dimension). There is no limit on the number of subscripts nor on their formation. The number of subscripts must equal the number of dimensions of the array declaration.	HAL makes use of subscripts for three purposes: I) to select (i.e., index or partition) data items from complex data types, arrays, and structures; II) to formulate types and arrays from component parts; and III) to modify the interpretation and usage of data quantities. Subscripting can be done in single line format or on separate subscript lines. Each level of dimensionality may be subscripted

1.0	Form of Language	
	PL/1	HAL
	Subscripts can take on any integer values between the declared lower and upper bounds. Upper and lower bounds may be expressions that are evaluated when storage for array is allocated.	separately. The division between structure subscripts and structure element subscripts is marked by a semicolon, that between array and array element by a colon, and the divisions between levels within the structure, within the array, and within the element (if more than one level -- matrices only) are marked by commas. Each subscript may select a single element or a partitioned range. Partitioning, subscripting to a smaller range rather than to a single indexed element, is a unique feature of HAL and is accomplished with the AT, TO, *, and # operators. All operate within a single subscript level. AT selects a certain number of components starting at a specified point; TO selects the range from one component to another; * represents the full range of the level; and # represents the last element in the range of the level. I. Data types including arrays, vectors, and matrices can be indexed to specify any of their scalar components or can be partitioned into any subset having the same data type. Matrices may be partitioned into vector components and arrays of two or three dimensions may be indexed to form arrays of 1 or 2 dimensions. Arrays of vectors and matrices are possible and may be indexed and partitioned on the array and

	Form of Language	
1.0	PL/1	HAL
1.3.5.1.1 Dimensions	1 to N Dimensions.	The maximum number of array dimensions is implementation dependent.
1.3.5.1.2 Classes	Subscripts can be any variable or expression that can be evaluated and converted to an integer including other subscripted variables. The lower limit can be specified and it can be negative.	Subscripts can be any scalar or integer variable or expression having a value >= 1. Subscripted subscripts are possible. In addition, the characters * and # are used to facilitate partitioning, * meaning all elements of that dimension and # the last element. TO and AT may be used to select subsets of a dimension (see 1.3.5.1 above).
1.3.5.1.3 Form	Non-integer variables or expressions are truncated to give an integer value.	Non-integer variables, literals, and expressions are rounded to the nearest integer.
1.3.5.1.4	Any data types can occur in array declarations. Elements of arrays may also be structures.	All data types except unarrayed integers and scalars may be subscripted. All types may be arrayed.
1.3.5.2 Qualification	A qualified name is a compound name, with its components separated by periods (.) and is usually used to avoid ambiguities in cases where a particular identifier or statement label (name) may be used within two or more different structures. Any name of a variable, array or structure can be qualified by preceding the name of	Structure names need not be declared with the QUALIFIED attribute unless a variable name or level name is used more than once. In that case, the structure name is written with the levels separated by periods (.), down to the individula variable name. Level names need not be used in referencing data from a structure with the NONQUALIFIED attribute.

1.0	Form of Language: PL/1	HAL
	the item with the names of one or more of the containing structures. The structure names must be sequenced from left to right in order of increasing level numbers. The sequence of names need not include all the containing structures but must have sufficient qualification to resolve all ambiguities. Structures can consist of variables and arrays. There is no maximum depth of qualified names.	
1.3.5.3 Qualification and Subscription	Indexing follows the same rules as ordinary indexing.	Structures may have more than one copy at each level of nesting, and up to 5 levels. They may contain any data type. Indexing is done in order, level by level and finally the array and vector, matrix, or string. Special separators (; and :) in addition to commas to divide between levels (structure subscripts; array subscripts: variable subscripts) of subscripting.
1.3.5.4 Numbering Conventions	Subscripts may take on any integer value between the declared bounds. Structures must start at level 1.	See 1.3.5.1.

1.0	Form of Language: PL/1	Form of Language: HAL
1.3.6 Scope of Names Based on or Relative to Program Structure		
1.3.6.1 System Names (Shared)	Storage for AUTOMATIC variable declarations is shared by subroutines not executing at the same time.	Variables may be shared by more than one program through a common data pool (COMPOOL). The COMPOOL may be either a group of declaration statements appearing before the program statements of each of the programs sharing data, or a set of these statements residing in a symbolic library and included before the program statements with the INCLUDE statement (see Sec. 4.5.2).

1.0	Form of Language: PL/1	Form of Language: HAL
1.3.6.2 Global Names (Shared)	It is possible to use the same name for different declarations of the same identifier through the EXTERNAL attribute. All external declarations for the same identifier are considered to be related to the same name; the scope of the name is the union of the scopes of all the external declarations for the identifier. Local names which are declared in outer procedures are global to inner procedures. Local names override global names.	Names are recognized at all lower levels unless they are redefined by a new DECLARE statement or excluded by an OUTER statement. An OUTER statement lists all the names from the containing blocks which are to be recognized. If the OUTER statement is used, names not mentioned in it are not recognized within the block in which it appears.
1.3.6.3 Local Names	Enclosed by BEGIN END pair. All the text of a begin block except the labels preceding the heading statement of the block is said to be contained in the block. All the text of a procedure except the entry names of the procedure is said to be contained in the procedure. That part of the text of a block B that is contained in block B but not contained in any other block contained in B is said to be internal to block B. The scope of a declaration of an identifier is defined as that block Bото which the declaration is internal, but it excludes from block B all contained blocks to which another declaration of the same identifier is internal. This definition of scope applies to all identifier declarations except those for entry names of external procedures.	Enclosed by a PROCEDURE, TASK, or FUNCTION block. These names are not recognized outside the block and may be excluded from a contained block as described immediately above (1.3.6.2). The name of the block (procedure, etc.) is recognized in the block which contains it. The scope of a label or block name may be extended beyond the block which contains it by being declared at a higher level.

1.0	Form of Language: PL/1	Form of Language: HAL
1.4 Definition and Usage of Other Basic Elements		
1.4.1 Operators		
1.4.1.1 Arithmetic Operators		
1.4.1.1.1 Scalar	+ (addition) - (subtraction) * (multiplication) / (division) ** (exponentiation)	+ - b (multiplication) / ** (single line format) Exponentiation is usually accomplished in the ordinary mathematical manner, by writing the exponents on exponent lines above the main line.
1.4.1.1.2 Non-Scalar	Element by Element Array Operations. Also applicable to structures. + (addition) - (subtraction) * (multiplication) / (division)	Vector: + - . (dot product) * (cross product) b (outer product of two vectors or vector scalar multiplication) / (division by scalar. Vectors may never appear in denominator) Vector operands must have same number of components except for outer product

	Form of Language	
	PL/1	HAL
		Matrix: + - T used as exponent (transpose) -1 used as exponent (inverse) b (multiplication by another matrix or by scalar or vector) / (by scalar only) Array: All operations permissible for arrayed data type may be performed with the entire arrays or portions of the arrays or between arrays and unarrayed variables. The operations are interpreted on a component by component basis and the arrayness (dimensionality) of the two arrays must be the same.

1.0	Form of Language: PL/1	HAL
1.4.1.2 Comparison Operators		
1.4.1.2.1 Binary	< or LT (less than) <= or LE (less than or equal) = (equal) > or GT (greater than) >= or GE (greater than or equal) ¬= or NE (not equal) ¬< or NL (not less than) ¬> or NG (not greater than)	< <= = > >= ¬= ¬< ¬>
1.4.1.2.2 Unary	>0 (positive and non-zero) <0 (negative) > = 0 (positive) < = 0 (negative or zero) ¬ = 0 (non-zero) 0 (zero)	same as PL/1
1.4.1.3 Boolean Operators	No explicit BOOLEAN data type but one bit bit strings suffice.	
1.4.1.3.1 Binary	& or AND | or OR	same as PL/1

1.0	Form of Language: PL/1	HAL
1.4.1.3.2 Unary	¬ or NOT	same as PL/1
1.4.1.4 Bit String Operators	The BOOL function also handles strings of information.	
1.4.1.4.1 Binary	& or AND | or OR	& or AND | or OR
1.4.1.4.2 Unary	¬ or NOT	¬ or NOT
1.4.1.5 Functional Modifiers	No	No
1.4.1.6 Others	The SUBSTR function handles textual and string operations. See Figure 1A. || or CAT (concatenate) → or PT (pointer)	|| or CAT
1.4.2 Delimiters	The delimiters are operators, parentheses, separators, and other delimiters.	The delimiters are operators, parentheses, separators, and other delimiters.
1.4.2.1 Brackets	( ) (enclose lists; specify information associated with keywords; delimits portions of computational expressions in conjunctions with operators or operands;) /* */ (comments) ' ' (encloses string constants and picture specifications)	( ) (many uses) /* */ (comments) ' ' (encloses string literals) DO END (groups) PROCEDURE CLOSE TASK CLOSE FUNCTION CLOSE PROGRAM CLOSE (blocks)

1.0	Form of Language	
	PL/1	HAL
	BEGIN END (blocks) DO END (groups)	
1.4.2.2 Separators	60 Character Set : (connects prefixes (label, conditional) to statements; also for bounds of array) ; (statement terminator) b (separates elements of statement) = (assignment)* , (separates elements of list) _ (break character) . (decimal binary point; connects elements of qualified name) % (prefixes macro statements executed by preprocessor) 48 Character Set .. (statement identifier terminator) ,. (statement terminator) BY TO (loop parameters) * also used as equal sign.	: (follows labels, used in subscripting) ; (statement terminator) b = (assignment) , (separates list elements) . (decimal point, connects elements of a qualified name) BY TO (loop parameters)

1.0	Form of Language: PL/1	Form of Language: HAL
1.4.3 Punctuation	The punctuation characters have both specific and general uses. In particular, the parentheses, colon, semicolon, comma, and period can be used to separate identifiers, constants or picture specifications. Identifiers, constants or picture specifications may not be immediately adjacent. They must be separated by either an operator, assignment mark, parenthesis, colon, semicolon, comma, period, blank or comment.	The punctuation characters have both specific and general uses. In particular, the parentheses, colon, semicolon, comma, and period can be used to separate identifiers and constants. Identifiers and constants may not be immediately adjacent. They must be separated by either an operator, assignment mark, parenthesis, colon, semicolon, comma, period, blank or comment. The semicolon, colon, and comma are used in subscripting.
1.4.4 Significance of Blanks	Blanks are optional between certain keywords (GO TO or GOTO), but they cannot be used between comparison operators. At least one blank must appear between a level number and its following identifier. Blanks can be used to separate identifiers, constants or picture specifications. Whenever one blank is used, any number can be used.	At least one blank must appear between a level number and its following identifier. Blanks can be used to separate identifiers and constants. Whenever one blank is used, any number can be used. Blanks are not optimal between keywords.
1.4.5 Literals	Hollerith or EBCDIC values are enclosed in single primes and optionally can be preceded by an integer in parentheses which specifies repetition. If quotes are desired in the strings they must appear as two immediately adjacent quote marks. Numeric values consist of signs + and - and of arabic numerals and at most one decimal point	Character strings are enclosed in apostrophes. For a repeated string, CHAR ( ) must precede the string, the number of repetitions being in the parentheses. Bit string literals can be specified with apostrophes preceded by BIN, OCT, or HEX. Repetition is possible with these also. Arithmetic literals are expressed as ordinary decimal numbers, signed or unsigned, and may have exponent powers of 2, 10, or 16 prefixed with B, D, or H.
1.4.6 Noise Words	No	

1.0	Form of Language: PL/1	HAL
1.5 Input Format		
1.5.1 Physical Input Format		
1.5.1.1 Linear		
1.5.1.1.1 Fixed (Columnar Restrictions)	No	No
1.5.1.1.2 String	There are no restrictions or rules about card columns or equivalent concepts. The entire program can be written as one continuous string from beginning to end, subject to rules of language. There can be more than one statement per line.	No
1.5.1.1.3 Combination	No	No
1.5.1.2 Non-Linear		
1.5.1.2.1 (Fixed) (Columnar Restrictions)	No	No
1.5.1.2.2 String	No, notation does not clearly resemble statement of problem or method of solution.	No

1.0	Form of Language: PL/1	Form of Language: HAL
1.5.1.2.3 Combination	No	HAL uses a three card-per-line format with the first column reserved for line-type specification C for comment, D for compiler directives, E for exponent, S for subscript, other characters and blanks for the main line. More or less than the 1 subscript and 1 exponent line per main line may be used as needed. Except for the restricted first column, HAL input may be in string format with statements separated by semicolons.
1.5.2 Conceptual Form		
1.5.2.1 Symbolic or Formal	No	Partially, because language is oriented to engineering/scientific problems.
1.5.2.2 English like	No	No
1.5.2.3 Pseudo English like	Reasonable but not complete formal symbolism. Language is close to ALGOL. Reference language is hardware language. Multipurpose language used for scientific, business and system programming.	Source program provides statements whose notation very closely resembles the problem definition (mathematical) and method of solution. Reference language is hardware language.

Structure of Program

2.0	PL/1	HAL
2.1 Statement Types		
2.1.1. Non-Executable Statements		
2.1.1.1 Declaratives	Explicit DECLARE statement parameter list statement label label of PROCEDURE or ENTRY statement Contextual Name in CALL statement, ON CONDITION, etc. Implicit If not explicit or contextual it is assumed implicit.	Explicit DECLARE statement OUTER statement (see 1.3.6.2) parameter list statement label label of PROCEDURE, FUNCTION, TASK, or PROGRAM statement Contextual Name in CALL statement Implicit If not explicit or contextual it is assumed implicit.
2.1.1.2 Compiler Directives	No	The INCLUDE statement (see 4.5.2). The REPLACE statement provides a means of replacing a name literally by a string of characters enclosed in single quotes. The substitution which does not appear in the listing is made whenever the name appears within the scope of the REPLACE statement (about the same as that of the DECLARE statement). REPLACE does not require a D in column one, though INCLUDE does.

Structure of Program

2.0	PL/1	HAL
2.1.1.3 Comments	A comment is defined as any string preceded by the double operators /* and terminated by */ except that the comment cannot contain the */ string. A comment can be used wherever a blank is permitted (except within a literal)	Same as PL/1, but with an alternative form: if column 1 contains a C, the line is a comment line.

Structure of Program

2.0	PL/1	HAL
2.1.2 Executable Statements		
2.1.2.1 Smallest Executable Statement	Simple statement of following types: assignment null (;) and of following form: ABLE = BAKER + 1;	Same as PL/1
2.1.2.2 Grouped Executable Statement		
2.1.2.2.1 Block Structure	A block is a sequence of statements that defines an area of a program. It is used to delimit the scope of a name and for control purposes. A program may consist of one or more blocks. Every statement must appear within a block. There are two kinds of blocks: begin blocks and procedure blocks. Begin blocks are delimited by BEGIN END statements. Procedure blocks are delimited by PROCEDURE END statements and must contain a label. Every begin block must be contained within some procedure block. Execution passes sequentially into and	There are several kinds of blocks in HAL which delimit name scope and are used for control purposes. A PROGRAM forms an outer block which may contain PROCEDURE, user-defined FUNCTION blocks, and TASK blocks (which permit real-time control). UPDATE blocks provide access to restricted variables.

2.0	PL/1	HAL
	out of a begin block. However, a procedure block must be invoked by execution of a statement in another block. The first procedure in a program to be executed is invoked automatically by the operating system. For System/360 implementations, this first procedure must be identified by specifying OPTIONS (MAIN) in the PROCEDURE statement. A procedure block may be invoked as a task, in which case it is executed concurrently with the invoking procedure.	
2.1.2.2.2 Others		

2.1.2.2.2.1 Compound Statements	Contain one or more other statements as a part of its statement body. These statements are the IF and ON statements. The final statement is a simple statement terminated with a semicolon. See the following:
	
	`IF A > B THEN A = B + C;`
	`ELSE GO TO LOOP_S;`
		Same as PL/1.

2.1.2.2.2.2 Groups	A group is a sequence of statements headed by a DO statement and terminated by a corresponding END statement. Labels are optional. There are simple and loop DO groups. For	A group is a sequence of statements headed by a DO statement and terminated by an END statement. Labels are optional. There are several DO statements: DO (simple form, followed by statements and END) DO WHILE (loops while condition is true)

	PL/1	HAL
	example: DO; A = A+1; B = B+1; C = C+1; END;	DO FOR (loops for a list of and/or incremented set of values of a control variable) DO CASE (chooses from all the statements between DO and END on the basis of the control variable value.) Some combinations are permitted.
2.1.2.3 Loops	Loops are handled by a DO statement or using an IF...THEN sequence. The iteration parameter follows the DO. See 4.3.3.	Loops are handled by the DO and IF statements.
2.1.2.4 Procedures, Functions or Sub-routines	There are two types of procedures, namely function procedures and sub-routine procedures, and each type can have multiple entry points. A procedure is considered a function if there is a specific result obtained when it is invoked. This result is indicated as part of the RETURN statement by which control is returned to the calling location. Functional procedures are normally used as operands in expressions. A subroutine procedure does not provide a value or part of the RETURN statement but specifies results by setting some of its parameters. It may only be invoked by a CALL statement or by a statement with a call option. Because a procedure may contain more than one RETURN statement it is possible to use it both as a function and as a sub-	There are procedures and functions as in PL/1.

2.0	PL/1	HAL
	routine procedure, although this is seldom done. A procedure that is not included in any other block is called an external procedure; a procedure included in some other block is called an internal procedure. See 4.3.1.2.	
2.1.2.5 Inclusion of Other Languages		
2.1.2.5.1 Assembly Language (AL)	No	No
2.1.2.5.2 Assembly Routines	No	No
2.1.2.5.3 Higher Order Language (HOL)	No	No
2.1.2.5.4 HOL-AL Communication	No	No
2.1.3 Intermingling Order for Non-Executable and Executable Statements	A program is composed of one or more external procedures. Thus, a program is a set of procedure blocks, each of which is completely nested and separate from the others. Statements and declarations can appear in any order except for the restrictions due to the placement of declarations to control the scope of variables. The concept of prologue is used to refer	Non-executable statements must be placed at the beginning of the block in which they occur.

2.0	Structure of Program	
	PL/1	HAL
	to computations that must be done at object time before executing the statements within a block. Thus the prologue must allocate storage or automatic variables and may need to evaluate expressions which define lengths, bounds, and iteration factors or to supply initial values to variables. Allocation and initialization cannot be circular. When a block is terminated the set of activities that must be performed before control can be transferred out of a block is called the epilogue.	
2.1.4 Operating System Interface	See Section 4.0 (Storage allocation, multitasking, interrupt conditions, input/output and compile time macro facilities).	See sections under 4.3.4 and 4.5.

2.0	Structure of Program PL/1	HAL
2.2 Statement Characteristics		
2.2.1 Methods of Delimiting		
2.2.1.1 Explicit and/or Contextual	The (;) indicates the end of executable statements and declarations. Labels are followed by colons, although the colon is also used for bounds on an array. Macro statements are preceded by the (%). A DO group is terminated by reaching an END statement with the same label or by the first END without a label.	The (;) indicates the end of executable statements and declarations. Labels are followed by colons. A DO group is terminated by reaching an END statement with the same label or by the first END without a label.
2.2.1.2 Implicit	Yes	Yes
2.2.2 Parameter Passage Required (Different Data Types)		
2.2.2.1 Call by Name		
2.2.2.1.1 Formal		
2.2.2.1.1.1 Input	No	No
2.2.2.1.1.2 Output	No	No

Structure of Program

2.0	PL/1	HAL
2.2.2.1.2 Calling		
2.2.2.1.2.1 Input	No	No
2.2.2.1.2.2 Output	No	No
2.2.2.2 Call by Value		
2.2.2.2.1 Formal		
2.2.2.2.1.1 Input	Constant, expression involving operators, expression in parentheses, a function reference containing arguments, variable whose data attributes are different from the data attributes declared for the parameter in an entry name attribute specification appearing in the invoking block, and controlled array or string associated with a simple parameter unless asterisk notation used.	Expressions are transmitted by value.
2.2.2.2.1.2 Output	No	No
2.2.2.2.2 Calling		
2.2.2.2.2.1 Input	Same as 2.2.2.2.1.1	Same as 2.2.2.2.1.1.
2.2.2.2.2.2 Output	No	No

Structure of Program

2.0	PL/1	HAL
2.2.2.3 Call by Address		
2.2.2.3.1 Formal		
2.2.2.3.1.1 Input	All cases not mentioned in 2.2.2.2.1.1	Same as PL/1.
2.2.2.3.1.2 Output	Same	Same
2.2.2.3.2 Calling		
2.2.2.3.2.1 Input	Same	Same
2.2.2.3.2.2 Output	Same	Same
2.2.3 Embedding	Statements can be embedded within other statements. Arithmetic expressions can contain functions, Boolean expressions, and concatenated strings. An expression can be used wherever its value can syntactically be used.	Same as PL/1.
2.2.4 Recursion	Procedures can be used recursively if they are so declared (RECURSIVE attribute).	No
2.2.5 Reentrant		
2.2.5.1 Serially Reusable	Yes	Yes

Structure of Program

2.0	PL/1	HAL
2.2.5.2 Reenterable	No	No
2.2.6 Pure Procedure	Implementation Specific.	Yes

Data Element Types, Groups and Operations

3.0	PL/1	HAL
3.1 Types of Data Elements		
3.1.1 Scalar		
3.1.1.1 Arithmetic	See 1.4.1.1.1.	
3.1.1.1.1 Integer	A special case of fixed point (zero fractional digits) is defined as integer.	Yes
3.1.1.1.2 Fixed Point (Mixed Number)	Decimal and Binary	Machine dependent.
3.1.1.1.3 Floating Point	Decimal and Binary	Machine dependent
3.1.1.1.4 Binary	Yes	No, binary strings may appear in arithmetic expressions, but they will automatically be converted to integers.
3.1.1.1.5 Octal	No	No, again, conversion is possible, but the conversion to binary must be done first with a built-in function.
3.1.1.1.6 Decimal	Yes	Yes
3.1.1.1.7 Hexadecimal	No	Same as Octal.

Data Element Types, Groups and Operations

3.0	PL/1	HAL
3.1.1.1.8 Multiprecision	User specifies the precision desired by defining data scale and total number of binary or decimal digits to be maintained for both fixed and floating point variables.	User can specify precision.
3.1.1.1.9 Other	No	No
3.1.1.2 Boolean	By use of a one bit string where: '1'B = TRUE '0'B = FALSE	Same as PL/1.
3.1.1.3 List or Pointer	POINTER and BASED variables which permit building of list structures.	No
3.1.1.4 Other	Pointer variable to some identifier provides the location of variable. Also, there are label, task, event and area variables. Area variables have been discontinued.	No
3.1.2 Non-Scalar		
3.1.2.1 Arithmetic	See 1.4.1.1.2.	
3.1.2.1.1 Vector	Defined by array declarations.	Yes

Data Element Types, Groups and Operations

3.0	PL/1	HAL
3.1.2.1.2 Matrix	Defined by array declarations	Yes
3.1.2.1.3 Complex	Yes	No
3.1.2.1.4 Other	No	No
3.1.2.2 Bit String	Yes	Yes
3.1.2.3 Text String	Yes	Yes
3.2 Groups of Data Elements		
3.2.1 Arrays	Yes	Yes, including arrays of vectors, matrices, and string variables.
3.2.2 Hierarchical Structures	Yes	Yes
3.2.3 Combinations of Above	Yes	Yes
3.2.4 Files	Yes	No, not as a separate data type. There is a FILE statement (see 4.5.1.1).

3.0	PL/1	HAL
3.3 Operations with Data Element Types and Groups		
3.3.1 Intermingling Rules For Mixed Data Types	Scalar expressions can contain any variable types except statement labels, area variables, task variables, and event variables. Only the comparison operators = and ¬= can appear with pointer data. Structure expressions and/or array expressions can be formed and their evaluation is done by evaluating the respective scalars. The most significant conversions are exactly what would be expected; namely decimals are converted to binary if both appear, fixed point operands are converted to floating point if both appear, and real numbers are converted to complex if both appear as operands of the same operator. Bit string operations can be performed on arithmetic data by converting them to bit strings. If comparisons are to be made among arithmetic, character strings, and bit strings, then the operand of lowest type is converted to the operand of highest type, where the priority is decreasing in the order just stated; i.e., bit string is the lowest priority. Only the operations of = and ¬= are defined if one of the operands is complex. Comparisons are actually of three types: algebraic, character (left to	Arithmetic expressions may contain any arithmetic variables and bit strings. Dimensionality of arrays, vectors, and matrices must be compatible and the operations must be legal (see section 1.4.1). Arrays can not be treated as vectors or matrices nor can vectors and matrices be treated as arrays, but conversion functions are provided for these and other conversions. Comparisons are similar to those in PL/1.

3.0	PL/1	HAL
	right pair-by-pair comparison of characters according to a given collating sequence), and bit (left to right comparison of the binary digits). The result of comparison is a bit string of length one; the value is '1' B if true and '0' B if false.	
3.3.2 Conversion Rules	Conversion is automatic across the assignment operator. See 3.3.1.	Most simple conversions are automatic within expressions and across the assignment operator (see 4.1.5). Conversions which are not apparent from context are possible with the conversion functions (see Figure 1B)
3.3.3 Alignment Rules	Alignment is automatic for arithmetic operations. There are implementation rules concerning truncation and range of values.	Alignment is automatic.
3.3.4 Precision and Computation Rules	The rules of precision are based on a complicated formula. The aim is to give a large enough result field for a fixed point operation (or the maximum size in the case of division) or a result field for floating point with the greater of the precisions of the two operands.	Precision is converted from single to double if one variable is double. Precision may also be specified using the @ operator in a subscript.

3.3.5 Precedence and Sequencing Rules

PL/1

Left to right scan with following operator precedence rules:

exponentiation	Highest
prefix +	↓
prefix −	↓
not	↓
multiplication, division	↓
infix +, concatenation (|)	↓
comparison operators	↓
and	↓
or	Lowest

If two or more operators of the highest priority (1st 4) in the same expression the order of priority of those operators is from right to left. For all others, if two or more operators of the same priority appear in the same expression. The order of priority is the normal left to right scan. Parentheses may be used to modify precedence. Nested parentheses evaluated innermost to outermost.

HAL

Left to right scan with following operator precedence rules:

Operation	Priority
exponentiation	6
matrix transpose (short form)	6
matrix inverse (short form)	6
scalar-scalar product	5
scalar-vector or vector-scalar product	5
scalar-matrix or matrix-scalar product	5
vector-matrix product	5
matrix-vector product	5
vector outer product	5
matrix-matrix product	5
vector cross product	4
vector inner (dot) product	3
scalar-scalar quotient	2
vector-scalar quotient	2
matrix-scalar quotient	2
scalar sum or difference	1
vector sum or difference	1
matrix sum or difference	1

Highest priority is 6. Same number means equal priority.

Data Element Types, Groups and Operations

3.0	PL/1	HAL
3.4 Accessibility of Data		
3.4.1 Hardware Defined	Bits and bytes can be processed with the bit string operators (AND, OR, NOT, CAT). There are also many built-in functions for string handling such as SUBSTR, BIT, CHAR and BOOL. Strings cannot be shifted. There are no machine register assignments or memory protection statements.	Bits and bytes can be processed with the bit string operators, with indexing, and with built-in functions. In addition, HAL possesses a built-in pseudo-variable (SUBBIT) which allows direct assignment of a bit string to the bit representation of a variable in memory.
3.4.2 Language Defined	All the variable types can be accessed by many of the language commands although some combinations are illegal. A command which is allowed to access a single variable can usually access an array or structure.	Same as PL/1.
3.5 Scope of Data	See 1.3.6.	See 1.3.6.

	Executable Statements	
4.0	PL/1	HAL
4.1 Assignment, Exchange and Computation Statements		
4.1.1 Data Element Types		
4.1.1.1 Scalar	Variable = expression; No exchange statements.	Same as PL/1.
4.1.1.1.1 Arithmetic	Fixed Point variables (Integer) Floating Point variables See 1.4.1.1.1. Multi-precision variables.	Integer and non-integer variables.
4.1.1.1.2 Boolean	Boolean operations are allowed and are evaluated for either a true or false condition. See 1.4.1.3. There are no limitations on the complexity of Boolean expressions. See also the BOOL built-in function.	Boolean operations are allowed and are evaluated for either a true or false condition. See 1.4.1.3. There are no limitations on the complexity of Boolean expressions.
4.1.1.1.3 Comparison	Comparison operations are allowed and are evaluated for either a true or false condition. See 1.4.1.2. There are no limitations on the complexity of comparison expressions. The three types of comparisons are: algebraic, character and bit.	Comparison operations are allowed and are evaluated for either a true or false condition. See 1.4.1.2. There are no limitations on the complexity of comparison expressions. The three types of comparisons are: algebraic, character and bit.

Executable Statements

4.0	PL/1	HAL
4.1.1.1.4 List or Pointer	Yes	No
4.1.1.1.5 Other	Concatenation operations are allowed and join character and/or bit strings. Different types of operations can be combined within the same expression and any combinations can be used. Label variables may be assigned.	Concatenation operations are allowed and join character and/or bit strings. Different types of operations can be combined within the same expression and any combinations can be used.
4.1.1.2 Non-Scalar	Variable = expression ; No exchange statements.	Same as PL/1.
4.1.1.2.1 Arithmetic	Complex variables. See 1.4.1.1.2. Multi-precision complex variables.	Matrix and vector assignments.
4.1.1.2.2 Bit String	Allows combining operands with the &, | and ||operators and they have the same function as in Boolean algebra.	Same as PL/1.
4.1.1.2.3 Text String	Bit string and concatenation operators are used on text strings.	Same as PL/1.
4.1.2 Arrays	Array and element operations, array and array operations and array and structure operations.	Like PL/1, but with sub-arrays and arrays of vectors and matrices. All possible subsets are subscriptable and may appear in assignment statements.
4.1.3 Hierarchical Structures	Structure and element operations, structure and structure operations, and structure assignment by name.	Same as PL/1.

	Executable Statements	
4.0	PL/1	HAL
4.1.4 Nested Assignment (Factoring)	With multiple variables but not expressions.	Same as PL/1.
4.1.5 Conversion Rules For Results	Mixed mode expressions are allowed and conversion rules for the different data types and operations are quite lengthy. Generally speaking, there is automatic data conversion in the operational expressions - Problem Data (bit string to character string, arithmetic to bit string, etc.), Locator Data (offset to pointer, pointer to offset). There is also automatic conversion by assignment for scalar expression operations (arithmetic, bit string, comparison, concatenation) and for non-scalar expression operations (arrays and structures).	Mixed mode expressions are allowed. Conversion functions are provided for conversions not clear from context alone. Arrays, matrices, vectors, structures, and combinations of these can all be interconverted with the built-in functions and subscripting. Conversions of bits to characters or integers, integers to scalars (non-integer), and integers and scalars to character strings are all automatic.

	Executable Statements	
4.0	PL/1	HAL
4.2 Textual Data Handling Statements		
4.2.1 Editing	No	No
4.2.2 Conversion	No	No
4.2.3 Sorting	No, dropped from language.	No
4.2.4 Comparison	No	All the comparison operators may be used. The collating sequence may be implementation specific.
4.3 Sequence Control and Decision Making Statements		
4.3.1 Unconditional Control Transfer		
4.3.1.1 No Return	By the GO TO statement which is followed by a statement label and terminated with semicolon. The STOP statement causes immediate termination of the major task and all sub-tasks and is followed by semicolon. Acts to return control to operating system. The EXIT statement causes immediate termination of the task that contains the statement and all tasks attached by this task. Normal sequencing is from one statement to next statement.	By the GO TO statement. The TERMINATE statement returns control to the executive.

4.0	PL/1	HAL
4.3.1.2 Return	The CALL statement invokes a procedure and causes control to be transferred to a specified entry point of the procedure. The form is: CALL entry name $(A_1,A_2,\ldots,A_N)$; where A_i are arguments. TASK, EVENT, PRIORITY are optional attributes to be discussed later. The RETURN; statement terminates execution of the procedure that contains the RETURN statement and returns control to the invoking procedure. It may optionally be followed by an expression. It may also return a value.	The CALL statement causes control to be transferred to the beginning of a procedure, task, or program. Control is returned by a RETURN statement or by the CLOSE statement. A function is envoked when its name appears. Control is returned by the RETURN statement.
4.3.2 Conditional Control Transfer	IF statement of form:	IF statement of form:
4.3.2.1 No Return	IF scalar expression THEN unit-1 [ELSE unit-2]. Unit = group, block True case = unit 1 False case = unit 2 or next statement. Unit 1 and Unit 2 can be labeled and can in turn be IF statements. Each ELSE clause is always associated with the innermost preceding IF which does not yet have an ELSE clause. An ELSE can be used with the null statement (;) for proper pairing.	IF L_c THEN S; or IF L_c THEN B ELSE S; where L_c denotes a logical condition or set of logical conditions, and S and B are statements (which may be labelled), where S may be another IF statement but B may not. B and/or S maybe DO...END groups.
4.3.2.2 Return	No	No

4.0	Executable Statements	
	PL/1	HAL
4.3.3 Loop or Index Control		
4.3.3.1 Loop Designation	Concise loop control with the loop option of DO statement of format: DO {pseudo-variable / variable} = specification[,specification]...; A specification has the following format: expression [TO expression2 [BY expression3] / BY expression3 [TO expression2]] [WHILE(expression4)]	The DO FOR statement provides a means of executing a DO...END group repetitively for a list of values of a control variable as well as for a logical condition. The list may contain a series of values and/or ranges of values. The general form is: DO FOR VAR = A,B,...C TO D BY E... WHILE L_c; where A,B,C,D,E may be scalar expressions and VAR is a scalar variable. "BY E" and "WHILE L_c" are optional.
4.3.3.2 Range of Loops	The DO block containing the given DO as its heading. In the DO WHILE option, there is no parameter but merely a termination condition indicating when to stop executing the range of the DO. Another option allows the variable to be a label, string, or complex variable, providing the last three items produce legal programs when used with the appropriate expression of the iteration.	The DO FOR block with the given DO as its heading. In the DO WHILE option, there is no parameter, but merely a termination condition indicating when to stop executing the range of the DO FOR.

	Executable Statements	
4.0	PL/1	HAL
4.3.3.3 Iteration Control Mechanism	Uses a DO expression, TO expression, a BY clause and a WHILE clause. The DO expression defines the initial values, the BY clause provides the increment and the TO expression provides the final value. The WHILE option is described above. If the BY clause is omitted then the expression (3) is assumed 1. If the TO expression is omitted, then the iteration is performed until terminated by either the WHILE clause or by some other statement within the range of the DO group. If both are omitted, there is a single execution of the DO group with the parameter having the value of the expression (1) which is the initial value. If the label option is used then the WHILE clause must be used. When the loop is terminated control is transferred to the statement immediately following the END of the DO group.	Uses a DO FOR expression, TO expression, a BY clause and a WHILE clause. The DO FOR expression defines the initial values, the BY clause provides the increment and the TO expression provides the final value. There may be a list of single values and/or TO BY sequences. The WHILE option is described above. If the BY clause is omitted then the expression is assumed 1. If the is omitted, then the iteration is performed until terminated by either the WHILE clause or by some other statement within the range of the DO group. If both are omitted, there is a single execution of the DO group. If both are omitted, there is a single execution of the DO group. When the loop is terminated control is transferred to the statement immediately following the END of the DO group.

Executable Statements

4.0	PL/1	HAL
4.3.4 Real Time Control		
4.3.4.1 Multitasking	Under the CALL statement (See 4.2.1.2) the TASK, EVENT, and PRIORITY specify that the called and calling procedures are to be executed asynchronously. The task is not a set of instructions but rather the execution of a set of instructions. There is always one major task and optionally available subtasks; each of the latter category can be named and the name can be used to refer to and set the priority of the task. A task can be suspended by the programmer until some particular point in the execution of another task has been reached; this specified point is known as an event and it can in fact be associated with the completion of a particular task.	Tasks are subroutines intended to be scheduled in real-time through an executive system. They may be initiated as procedures are with the CALL statement, or alternatively with the SCHEDULE statement which specifies priority to choose among tasks scheduled for the same time. Tasks may be suspended and reactivated with the WAIT statement, and priorities may be changed with the PRIOCHANGE statement.
4.3.4.2 Scheduling and Dispatching	The WAIT statement causes the task in which it is executed to be suspended until the condition EVENT (event-name) equals a true condition. All the event names listed, or a number equal to the value of the optional expression (if used), must satisfy the condition in order for the task issuing the WAIT statement to be allowed to resume. The DELAY statement causes execution of the controlling task to be suspended for n milliseconds, where n is the value of the expression shown in the format. Execution resumes after n	The SCHEDULE statement is used to request initiation of a program or task based on three criteria: a) at a specific time; b) in an incremental time; c) on events or combinations of events. The WAIT statement is used by an active program or task to suspend and reactivate itself based on three criteria: a) a specific time; b) an incremental time; c) a particular event or combination of events. Programs and tasks may be scheduled by the occurrence of events or combinations of events. An event

4.0	PL/1	HAL
	milliseconds only if the controlling task is of sufficently high priority to be selected in preference to all other ready tasks.	is a programmer-named condition and can be stimulated only by the execution of the SIGNAL statement. Event-variables must be declared using DECLARE statements. The format is similar to that for data declarations. If the attribute LATCHED is provided, the event-variable will hold its signalled value; i.e., if signalled on, it will remain on. If LATCHED is not specified, the event-variable when signalled on, will remain on only for a short interval of time. The time interval is implementation dependent. The SIGNAL statement is used to cause the occurrence of an event. The specific effect depends upon whether the event-variable has the attribute LATCHED. The PRIOCHANGE statement is used to change the priority of a task or program.

4.0	PL/1	HAL
4.3.4.3 Storage Allocation	See 4.5.4	See 4.5.4
4.3.4.3.1 Static		
4.3.4.3.2 Dynamic		
4.3.4.4 Access Restrictions		
4.3.4.4.1 Memory	The UNLOCK statement does not cause data to be transmitted, but is used to unlock a record in an EXCLUSIVE DIRECT UPDATE file. When a record is read from a DIRECT UPDATE file, it may subsequently be rewritten or deleted to complete the updating operations; if the file is EXCLUSIVE, the read operation automatically locks the record to prevent interference by other tasks during this process. The UNLOCK statement makes the specified record available to tasks other than that for which the locking READ statement was issued; it therefore provides an alternative means of completing the updating operation.	HAL provides features to control the sharing of variables in order to prevent conflicts in their utilization. These features include the attribute LOCKTYPE to designate shared variables and an update block of statements in which shared variables may be changed in a controlled environment. Although the approach taken is basically implemented in software, it does depend on the ability to perform an "uninterruptable" instruction similar to the Test and Set instruction available on IBM System 36 computers. Read and write accesses of shared variables are confined to identified update blocks. The compiler assigns a locking control variable to each variable declared with the LOCKTYPE attribute. The value of the "lock" is examined at run-time and only consistent (i.e., safe) accesses are permitted. There are two locktypes: LOCKTYPE(1) data is read and write-locked when being accessed. LOCKTYPE(2) data is write-locked only, permitting

4.0	PL/1	HAL
		reads anytime. All locked variables to be assigned new values (i.e., updated) must appear within update-blocks. LOCKTYPE(2) variables which are to be read only need not be confined to these blocks. Execution of the UPDATE statement attempts "to lock" all shared variables within the block. A variable to be assigned will be write-locked, variables to read only will be read-locked. Once locks are established they are not opened until execution of the CLOSE statement at the end of the block. If all desired locks cannot be established at the UPDATE statement because one or more of the shared variables are not available (i.e., they are already locked elsewhere), the current program or task will be stalled (placed in "wait" by the executive) until all variables become available. The general use of COMPOOL data (see sec. 1.3.6.1) within programs may be restricted by attaching access rights to the DECLARE statement within the COMPOOL. Programs are identified by number and permitted to access only those variables which have been declared with corresponding identification numbers. An illegal reference to a COMPOOL variable will prevent successful compilation of the problem. The attribute EXCLUSIVE may be applied to programs, procedures, functions and tasks which are intended to be executed serially. It prohibits reentrancy.

4.0	Executable Statements	
	PL/1	HAL
4.3.4.4.2 Registers	No	No
4.3.4.4.3 Interrupts	No	No
4.3.4.5 Interrupt Handling	See 4.3.5.	An interrupt may be caused by a system- or user-defined error condition (see 4.3.5) or by a SIGNAL or WAIT statement or by a job swap based on priority (SCHEDULE statement) (see 4.3.4.2).

4.0	PL/1	HAL
4.3.5 Error Condition and Program Checking	Users can override standard error checking actions by using the ON-CONDITIONS statement. The programmer may specify some action to take place and in that case it is considered as a procedure internal to the block in which it appears. The on-unit to be performed when the conditions occurs is either an unlabeled simple statement (other than BEGIN, DO, END, RETURN, FORMAT, PROCEDURE, or DECLARE) or an unlabeled begin block. If SNAP is specified, a calling trace is listed when the given condition occurs. There are specific detailed rules about the scope of the ON statement. A condition raised during execution results in an interrupt if and only if the condition is enabled at the point where it is raised. Most conditions are enabled by default, and the remainder are disabled by default. For several, the enabling or disabling may be controlled by the use of condition prefixes. An interruption for most error conditions of a general type will occur whether or not an ON statement has been executed; the ON statement merely determines the action to be taken when the condition arises, but it has nothing to do with allowing or preventing an interruption to occur. However, the programmer can actually control certain interruptions through the	Programmer-defined error conditions can be generated with the SEND statement. Users can respond to user-defined error conditions or can override the system error-checking actions by using the ON statement. The system error condition or conditions are specified in this statement with an implementation-specific code. If one of the specified errors occurrs within the scope of the statement, control is transferred to the labelled statement specified, or to the system. The part of the ON statement's containing block which follows the statement forms the scope of the statement.

4.0	PL/1	HAL
	use of condition prefixes. An enabling condition prefix is a list of condition names, enclosed in parentheses, and prefixed to a statement with a colon that precedes the label. Thus the user can write (SIZE, SUBSCRIPTRANGE):LABEL: executable statements;. By preceding certain condition names with the letters NO, the condition is disabled from causing an interrupt. If the condition name is prefixed to any statement other than a PROCEDURE or BEGIN statement, the condition is enabled (or disabled) only through the execution of that single statement. If it is prefixed to an IF statement, its scope is only through the evaluation of the expression in the IF clause. If a condition name is prefixed to a DO statement, its scope is only through the DO statement itself. If the condition name is prefixed to a PROCEDURE or BEGIN statement, its scope is through the entire block including all nested blocks except for any statements that lie within the scope of another condition prefix with a different specification for the same condition. Unlike the scope in an ON statement, the scope of a condition prefix does not extend to a block that is invoked remotely. The REVERT command has essentially the effect of canceling an ON statement once the latter has been actually	

4.0	PL/1	HAL
	executed, assuming that they were internal to the same block. It also reactivates the most recent ON statement in the containing block. It is possible to simulate the existence of one of the interrupts through the use of the SIGNAL statement, which causes the same action as if the specified condition had actually occurred. These conditions are classified as follows: Computational (Data Handling) CONVERSION, NOCONVERSION FIXEDOVERFLOW, NOFIXEDOVERFLOW OVERFLOW, NOOVERFLOW SIZE, NOSIZE UNDERFLOW, NOUNDERFLOW ZERODIVIDE, NOZERODIVIDE Input/Output (Data Transmission) ENDFILE (FILE NAME) ENDPAGE KEY NAME PENDING RECORD TRANSMIT UNDEFINEDFILE Program Checkout (Debugging) CHECK (IDENTIFIER LIST) SUBSCRIPTRANGE STRING RANGE	

4.0	Executable Statements	
	PL/1	HAL
	List Processing AREA Programmer Named CONDITION (IDENTIFIER) System Action FINISH ERROR	

4.0	Executable Statements	
	PL/1	HAL
4.4 Symbolic Data Handling		
4.4.1 Algebraic Expression Manipulation	No, but there exists a PL/1 - FORMAC interpreter which is akin to the FORTRAN - FORMAC interpreter.	No
4.4.2 List Handling Statements	No, but POINTER variables and BASED variables allow the user to define his own list structure. Operations are allowed through assignment, by SET (ADDR, NULL, NULLO, EMPTY). The symbol → is used for list structure that may be doubly linked.	No
4.4.3 String Handling Statements	No, but there is a character string data type and these are handled with the assignment or other statement. The AND, OR, NOT and ||(concatenate) operators are legal. There is also a number of built-in string functions (BIT, CHAR, SUBSTR, etc.).	No, but there is a character string data type and these are handled with the assignment or other statement. The AND, OR, NOT and ||(concatenate) operators are legal. There is also a number of built-in string functions (BIT, CHARACTER, INDEX, LJUST, etc.).
4.4.4 Pattern Handling Statements	No, but a small capability is provided by the INDEX function.	No

Executable Statements

4.0	PL/1	HAL
4.5 Interaction with Operating System and/or Environment		
4.5.1 Input/Output Statements		
4.5.1.1 File Initialization/Processing/Termination	PL/1 allows for two different kinds of data transmission, namely stream- and record-oriented. The verbs GET and PUT are used for input and output of data items in the stream, while the statements READ and WRITE do similar things for the record-oriented data. In the stream-oriented case, the data is considered to be a continuous stream of data items in character form, and an assignment must be made from the stream to the variables or vice versa. With record-oriented transmission, the data set is considered to consist of a collection of physically separate records, each of which consists of one or more data items in an encoded form; each record is transmitted as an entity directly without any conversion. There are three types of stream-oriented transmission, namely list-directed, data-directed and edit-directed. In each case, the user supplies the file name and the list of variable names involved (which is called a data list). For edit-directed data, the format of each data item must be given. In several cases there are default conditions so the user need not always supply this information explicitly. In the list-directed transmission, the data items in the	The HAL input-output statements provide for the filing, retrieval, reading and writing of data to and from external storage media. Filing is record-oriented in that a file statement causes a single record to be transmitted to or from a storage device; transmission is direct without any conversions. Reading and writing are stream-oriented in that data is considered to be a continuous stream of characters; conversions may occur during transmission. There are four basic I/O statements in HAL: two read statements: READ and READALL. The READ statement causes data, in standard formats from an external source, to be assigned to a list of variables. The READALL statement allows data in non-standard form to be assigned to HAL character-string variables. This is accomplished by not defining fields of data but accepting all characters encountered in the input stream, including blanks, commas, semi-colons and apostrophes. The WRITE statement causes the transmission of data to an external device. Data items transmitted are the character string representations, in standard formats, of values of

PL/1	HAL
stream are always written as arithmetic or string constants. The user provides in the GET or PUT statements a list of variables to which the data items are to be assigned or to be output from in sequence; the variables in the data list are separated either by commas or blanks when used as input, while on output the blanks are supplied automatically between items. The PUT statement also allows the user to write an expression in his list and the output is the value obtained by evaluating the expression. In the data-directed form of transmission, the data list need not appear in the GET statement because the strea stream is in the form of a series of assignment statements that specifies each variable name and the value assigned to it. Note that the last data item is followed by a semicolon which is used to delimit the number of items obtained by a single GET statement. On output, the data list must be written to specify which data items are to be written into the stream. On input, the assignments can be separated by commas or by blanks. On output, blanks are supplied and the semicolon is written after the last item specified in the data list. In both the data-directed and list-directed cases, there are various rules about what types of data may appear in the data list, when and how	HAL expressions. The FILE statement has the appearance of an assignment statement and may be used for both filing and retrieving data depending upon which side of the = sign FILE appears. It handles data in records of a size which may be implementation dependent. Each record must be assigned to or from a single array or structure.

4.0	PL/1	HAL
	subscripts can be used, and when subscripts are evaluated. The edit-directed transmission allows the user to control the format by providing information about things such as precision, strings, conver-ion sion, etc. In addition, the user can control pagination and lines through the use of the PUT statement. The LINE option causes the data to be written on a new line; the PAGE option allows the user to start a new page; the expression in the LINE option essentially controls which line is used, i.e., which new line the data will start on. The SKIP option also allows the user to control the start of a new line and to indicate how many lines are to be skipped; however, it also permits the user to overprint a particular line. It is possible to use the input/output statements for an internal character string; this is done by using the STRING option, which allows the user to obtain information from an internal character string and place data there. Although the string option can be used with any of the three types of stream-oriented transmission, it is usually most practical in association with a format list since individual items in the string need not be separated by commas or blanks. Each READ and WRITE statement transmits a single logical record between the external medium and the variable specified, without any conversions.	

4.0	PL/1	HAL
	The variable specified must be a "level-1" item and normally contains several data items or arrays. If the file specified in a READ or WRITE statement is not open when the command is given, it is opened automatically. The variable associated with the words INTO and FROM in the READ and WRITE, respectively, specifies the variable in internal storage into which or from which the record is to be read or written. In the READ statement, the SET option places a record in a buffer and assigns a pointer variable as its identification so that a based variable can be subsequently referred to via the pointer value. The IGNORE option may be specified for SEQUENTIAL INPUT and SEQUENTIAL UPDATE FILES. It controls the number of records that are skipped. The KEY option must appear if the file is DIRECT; the expression is converted to a character string but it determines which record is read. The KEYTO option can be given only if the file is SEQUENTIAL and keyed; it specifies that the key of the record is to be copied onto the string variable. The EVENT option allows processing to continue while the record is being read or ignored. The NOLOCK option prevents the statement from causing a record on an EXCLUSIVE file from being locked against access by other tasks.	

4.0	PL/1	HAL
	In the WRITE command, the KEYFROM option is converted to a character string and attached to the record as a key. A file can be INPUT, OUTPUT, or UPDATE. The REWRITE statement can be used for the UPDATE and serves the purpose of replacing an existing record in the data set in the file involved. The other key words provide options similar to those in the WRITE command. The OPEN statement causes the opening of a file and provides a number of additional file characteristics beyond those shown in the file description. The IDENT option associates the identifying user label on an input file with the variable given as the argument; for output, the argument is an expression which is evaluated and converted to a character string which is placed as a header label. If an input file is a BACKWARDS file, the label will be a conversion of the specified expression to a character string which identifies the data set associated with the file; if this option does not appear, the file name is taken as the identifier. The other options are either self-explanatory or are discussed under the file description. The CLOSE statement dissociates the named file from the data set with which it was associated by opening, and it also dissociates all the	

4.0	PL/1	HAL
	attributes declared for it in the original opening of the file. However attributes for that file which are explicitly given in a DECLARE statement remain in effect. The argument in the IDENT option essentially serves as a trailer label. The DELETE statement removes a record from a DIRECT UPDATE file. The expression associated with the KEY identifies the record to be deleted. The DELETE statement can cause implicit opening of a file. The UNLOCK statement makes accessible a record which would otherwise be inaccessible as a result of the READ statement accessing it from an EXCLUSIVE file. The LOCATE statement applies to BUFFERED OUTPUT files and allows a record to be created in buffer storage and later written out. The SET option specifies a POINTER variable which is to be set to identify the variable in the buffer. The DISPLAY statement causes a message to be displayed to the machine operator; a response may be requested by using the REPLY option. If the EVENT option is used, execution of subsequent statements will continue before the reply is completed.	

4.0	PL/1	HAL
4.5.1.2 File Positioning and Handling	These operations are automatically performed or handled by use of the OPEN, CLOSE statements. In addition LOCATE, DELETE, UNLOCK provide the other options.	External data media, either providing input information to a HAL program or accepting output data, are treated as two-dimensional devices. Data occupies a grid consisting of horizontal lines with each line being made up of column positions; for example, a deck of punched cards where each card is a line, or a 132-column high speed printer. The "read mechanism" or "write mechanism" is located at some point on this two-dimensional grid, and moves in a conventional way along each line and from line to line as reading or writing takes place. Read- and write-control functions are used to move the "read mechanism" or "write mechanism" to any reachable location desired in readiness for reading or writing. The definition of "reachable" varies depending on the physical device involved. The "read mechanism" is located on the two-dimensional grid by the read-control functions SKIP, TAB and COLUMN. A READ statement without these functions will always begin on column 1 of the next line, and will then read consecutive data fields, line after line, until all variables have been assigned values, unless interrupted by a semicolon terminated a data field. The SKIP read-control function controls the vertical position of the "read mechanism"; that is, it controls which line is

4.0	PL/1	HAL
		next to be read. The form is SKIP(N), where N ≥ 0 A SKIP(0) in the middle or at the end of a list of variable names has no effect. A SKIP(0) before the first variable name in a READ statement causes reading to continue on the same line as that last read by the previous READ statement. There is no relocation of the horizontal position of the "read-mechanism" during the skips. The TAB and COLUMN read-control functions control the horizontal position of the "read mechanism", at which reading is to start or resume. The TAB function moves the "read mechanism" left or right by the specified number of columns. Its form is TAB(N), where N<0: move to left; N=0: no effect; N>0: move to right. COLUMN sets number of next column. The "write mechanism" is located in the two-dimensional grid by the write-control functions LINE, SKIP, PAGE, TAB and COLUMN. TAB and COLUMN have the same effect as when used as read control functions. LINE, PAGE, and SKIP control the vertical position of the "write mechanism". The PAGE

4.0	PL/1	HAL
		function is of the form PAGE(N), N≥0 and causes the printer to advance N pages, remaining on the same line relative to the head of the page. (Each page has 58 lines.)

4.0	PL/1	HAL
4.5.2 Library Reference Statements	Built in functions (see figure 1A). Function names can be used as identifiers but they must be explicitly declared.	Built-in functions and the INCLUDE statement. The INCLUDE statement modifies a program at compile time to include some lines of code stored in the symbolic library. One use of this statement is to include the shared data declarations which form a COMPOOL (see 1.3.6.1).
4.5.3 Debugging Statements	See 5.3.5. The SNAP and On statements	ON and SEND statements (see 4.3.5).
4.5.4 Storage and Segmentation Allicaton Statements		
4.5.4.1 Static	There are only two actual commands dealing with storage allocation, namely ALLOCATE and FREE. There are four classes of storage which are controlled by the declarations. In order to understand the meaning of the statement, it is necessary to discuss the storage categories. The four storage classes are static, automatic, controlled, and based. Static storage is assigned before first entry to the program and remains in effect throughout the life of the program.	Static storage, specified by the STATIC attribute in the declaration statement, is assigned before the first entry to the programs and remains in effect throughout the life of the program.
4.5.4.2 Dynamic	Automatic storage is assigned as object time upon entry to the block in which it is declared and released upon exit from that block. The dimensions can be variables or expressions. Controlled and based storage are under	Automatic storage, specified by the AUTOMATIC attribute in the declaration statement, is assigned upon entry to the block which it is declared in and released upon exit from that block. There are no specific segmenting instructions in HAL.

4.0	PL/1	HAL
	programmer control. Variables declar declared as CONTROLLED and BASED can and must have storage assigned and released by the ALLOCATE and FREE statements, respectively. The ALLOCATE command essentially serves the purpose of placing something on top of a pushdown stack, while the FREE command performs the popup function. References to a stacked controlled variable always refer to the most recent allocation, whereas all current allocations for a based variable can be obtained by a pointer value. An option of the ALLOCATE statement indicates a BIT, CHARACTER, or INITIAL attribute, where the first two can only appear with identifiers of that type. Since bounds or lengths can be specified in the ALLOCATE statement, they override any similar information which might be included in a DECLARE statement; if no bound or length is specified in the ALLOCATE statement, it must be specified in a DECLARE statement. When an identifier is allocated, the initial values will be assigned if the identifier has that attribute. To ascertain whether or not storage has been allocated for a particular identifier, the built-in function ALLOCATION may be used. Both controlled and based variables can be specified in the same ALLOCATE and FREE statements. In another option of the ALLOCATE statement, there is no pushdown list, and any genera-	

4.0	PL/1	HAL
	tion of the based variable can be referenced through a pointer variable. The SET clause indicates the pointer variable that is to receive the pointer value identifying the particular value of the variable for which storage is to be allocated. If the IN clause appears in the ALLOCATE statement, storage will be allocated in the named area for the based variable; if that clause is omitted, space will be allocated in systems storage. For based variables, all characteristics must be specified in the declaration and cannot be included in the ALLOCATE statement. In the use of the FREE statement, if a specific pointer qualification is not given for the based variable, then the pointer declared with the based variable will be used. In the first mentioned option a CONTROLLED variable must be used. There are no specific segmenting instructions in PL/1.	
4.5.5 Operating System and Machine Dependent Statements	Besides the input/output described previously there is the creation and execution of tasks. In the CALL statement, the TASK, EVENT, and PRIORITY options specify that the	In addition to input/output, the real-time control statements (see 4.3.4), the error condition statements (see 4.3.5), and the null statement (see 2.1.2.1).

4.0	PL/1	HAL
	called and calling procedures are to be executed asynchronously. The task is not a set of instructions but rather the execution of a set of instructions. There is always one major task and optionally available subtasks; each of the latter category can be named and the name can be used to refer to and set the priority of the task. A task can be suspended by the programmer until some particular point in the execution of another task has been reached; this specified point is known as an event and it can in fact be associated with the completion of a particular task. The WAIT statement causes the task in which it is executed to be suspended until the condition EVENT (event-name) is satisfied. All the event names listed, or a number equal to the value of the optional expression must satisfy the condition in order for the task issuing the WAIT statement to be allowed to resume. The DELAY statement causes execution of the controlling task to be suspended for n milliseconds, where n is the value of the expression shown in the format. Execution resumes after n milliseconds only if the controlling task is of sufficiently high priority to be selected in preference to all other ready tasks.	

	Executable Statements	
4.0	PL/1	HAL
	The STOP statement causes immediate termination of the major task and all subtasks. The EXIT statement terminates the task containing the EXIT statement and all tasks attached by this task; hence if the EXIT statement occurs in a major task, it is equivalent to a STOP statement. The Null statement causes no action and has no effect on sequential operation. It can be used with an ELSE to obtain the desired pairing in an IF statement.	
4.5.6 Others	No	No

Non-executable statements

5.0	PL/1	HAL
5.1 Data Declarations		
5.1.1 Data Element Type Declarations		
5.1.1.1 Constants		
5.1.1.1.1 Arithmetic		
5.1.1.1.1.1 Hexadecimal	No	HEX(K)'HHH...' where (K) is optional and gives number of repetitions. H is a hexadecimal digit (0 to F).
5.1.1.1.1.2 Decimal	Fixed Point (Integer) Floating Point	DEC(K)'DDD...' or integer or fixed point or floating point with decimal, hexadecimal, octal, or binary exponent.
5.1.1.1.1.3 Octal	No	OCT(K)'OOO...'
5.1.1.1.1.4 Binary	Fixed Point (Integer)	BIN(K)'BBB...'
5.1.1.1.2 Boolean	'1'B = TRUE '0'B = FALSE	BIN'1' = TRUE=ON BIN'0' = FALSE=OFF
5.1.1.1.3 Textual	'CCC...' or (K)'CCC...' where C = EBCDIC character string and K = repetitions of character string enclosed in quotes	'CCC...' or CHAR(K)'CCC...'

5.0	Non-Executable Statements PL/1	HAL
5.1.1.1.4 Other	'bbb...'B where B b = binary digit or (K)'BBB...' where K = repetitions of bit string enclosed in quotes.	No
5.1.1.2 Variables	The DECLARE statement is the principal means of specifying the attributes of a name. A name used in a program need not always appear in a DECLARE statement; its attributes often can be determined by context. If the attributes are not specifically declared and if they cannot be determined by context then default rules are applied. The combination of default rules and context determination can make it unnecessary, in some cases, to use a DECLARE statement. DECLARE statements are always needed for fixed-point decimal and floating-point binary variables, character- and bit-string variables, label variables, arrays and structures, static, controlled, and based variables, offset variables, and all data with the PICTURE attribute. An ENTRY declaration must be made in a DECLARE statement for the name of any function that returns a value with attributes different from the default attributes that would be assumed for the name -- FIXED	Names may be implicitly declared if the data type is indicated with exponent line notation above the variable name: a bar(-) for vectors, star (*) for matrices, period (.) for bit strings, and comma (,) for character strings, no notation means scalar. Default dimensions are applied: (3) for vectors, (3x3) for matrices, (1 bit) for bit strings, (8 characters) for character strings. The DECLARE statement allows the user to specify precision, STATIC or AUTOMATIC storage allocation (see 4.5.4) dimensions other than default dimensions, initialization with the INITIAL or CONSTANT attributes (CONSTANT prohibits reassignment), LOCKTYPE (see 4.3.4.4.1), and DENSE or ALIGNED (see 5.1.3.2).

5.0	PL/1	HAL
	BINARY(15) if the first letter of the name is I through N; otherwise, DECIMAL FLOAT(6). (The default precisions are those defined for System/360 implementations.) An ENTRY declaration also must be made if arguments and parameters do not match exactly, as may be the case when constants are passed as arguments. DECLARE statements may also be an important part of the documentation of a program; consequently, programmers may make liberal use of declarations, even when default attributes apply or when a contextual declaration is possible. Because there are no restrictions on the number of DECLARE statements, different DECLARE statements can be used for different groups of names.	

Non-executable statements

5.0	PL/1	HAL
5.1.1.2.1 Arithmetic	The FIXED, FLOAT, COMPLEX attributes and number base attributes of DECIMAL and BINARY. The precision is specified in fixed and floating format by the total digits (bits) in the word or for fixed format, the number of fractional digits.	Declare statements are needed for integers, for precision other than normal single precision determined by hardware, for vectors of other than 3 components, for matrices other than 3 by 3.
5.1.1.2.2 Boolean	Booleans are declared as BIT(1 BIT CHAR STRING DEFINED).	Booleans are declared as BIT(1), a one bit bit string.
5.1.1.2.3 Textual	One must specify the length in characters or use the VARYING attribute to specify the maximum length.	Character strings are declared as CHARACTER(length) and may have the attribute VARYING, so that they take on the length of the string assigned to them up to the maximum length specified by (length) above.
5.1.1.2.4 Other	The attribute (BIT(W)) where W = the length in bits. The VARYING attribute is also applicable.	Bit strings may be declared. They are not arithmetic, being converted to integers if used in an arithmetic expression. They may be compared and may be operated on with the OR, AND, NOT and CAT operators.

Non-executable statements

5.0	PL/1	HAL
5.1.1.3 Presetting of Declarations	Yes, INITIAL attribute used	This is accomplished with the use of the INITIAL option followed by a value or list of values in parentheses, the number of list elements being equal to the number of components of the variable. # may be used to indicate repetition.
5.1.1.4 Nesting (Factoring) of Declarations	Yes	This is possible and common attributes need only be given once, at the beginning of the list of variable names.
5.1.1.5 Default options	Yes	Only a minimum of information need be specified. Some of the defaults have been mentioned above. Other defaults are static and aligned.
5.1.1.6 Numbering Conventions	No	No

Non-executable statements

5.0	PL/1	HAL
5.1.2 Group Type Declarations		
5.1.2.1 Array Declarations	Yes, the array is named and its elements are referenced by subscripting the array name. There is no limit concerning the number of dimensions in the array. Subscripts may take on any integer value between the specified upper and lower bounds. These bounds can be expressions that are evaluated when array storage is allocated. Any variable data type and STRUCTURE's can be array elements. PL/1 allows the concept of an array cross section and uses the * to denote it.	The arrays must be declared as ARRAY followed by the dimensions in parentheses and by the element type if integer or non-scalar.
5.1.2.2 Hierarchical Structures	Yes, the tree oriented type STRUCTURE declaration. This allows reference to data by heading and subheadings. Elements of the hierarchy can be variables and arrays. Level numbers are specified with structure names. A qualified name is an elementary name or minor structure name that is made unique by qualifying it with one or more names at a higher level. These different names are connected by the period (.). There is no limit to the depth of qualified names. The ALIGNED, UNALIGNED attributes are used to specify the positioning in storage of data elements to influence speed of access or storage economy. The LIKE attribute is used to indicate that the name being declared is to be given the same	Structures may have up to five levels. Level numbers precede the major and minor structure names and the variable names in the declaration statement. When all the names associated with a structure are unique the data type names and the minor structure names may be referred to individually without ambiguity. Under these conditions the major structure may be given the attribute NON-QUALIFIED, i.e., its names need no further qualification. However, the names within a structure need not be unique. It is permissible to use some or all of the lower level names in several minor structures or in another major

5.0	PL/1	HAL
	structuring as the name following the keyword LIKE. The DEFINED attribute specifies that the named data element, structure or array is to occupy the same storage area as that assigned to other data.	structure declared in the same part of the program. In this case, the structure must be declared as QUALIFIED. Multiple copies of major and/or minor structures may be declared by including a dimension in the DECLARE statement after the structure name.
5.1.2.3 Procedure, Function, and Subroutine Declarations	Yes, of form LABEL: PROCEDURE or LABEL: PROCEDURE ($P_1 \ldots P_N$);. Procedures are invoked by CALL LABEL ($X_1 \ldots X_N$);. Passed parameters can be variables, arrays, structures, expressions, literals, procedure and statement labels. Formal and calling parameters must agree. Multiple entrances are available through ENTRY attribute. Transfer of control back to calling procedure is via RETURN; (within body) or END; (end of procedure body). Procedures can be nested. They may also be defined as RECURSIVE. Functions are the same as procedures except they are terminated with -RETURN(EXPRESSION). They are invoked by using the function name as in Fortran.	These should be declared at the beginning of the program block in which they appear if they are referenced before they are defined. Declarations are simple as: DECLARE A PROCEDURE or DECLARE B FUNCTION VECTOR
5.1.2.4 Presetting	Yes, via INITIAL attribute and iteration is allowed for identical values.	Arrays and structures may be initialized: arrays with a list, and structures by initializing the variables in the structure individually. In structures with multiple copies, a variable may be initialized identically or uniquely in each copy.

5.0	PL/1	HAL
5.1.2.5 Nesting (Factoring) of Declarations	Yes, but it is permitted only in the DECLARE statement.	Array declarations may be factored, but structures may not.
5.1.2.6 Default Options	Yes, however there are a few cases where there is no default option: string attributes.	There are no default dimensions for arrays. Structures default to NONQUALIFIED if not specified to QUALIFIED. The usual defaults apply for the variables which make up arrays and structures.

Non-executable statements

5.0	PL/1	HAL
5.1.2.7 Numbering Conventions	No	No
5.1.3 Interaction with Operating System and/or Environment		
5.1.3.1 File Declarations	Standard system I/O files need not be declared or referenced. Others must - see 4.5.1.1.	Files need not be declared as such. Filed data, when accessed, is assigned directly to arrays or structures whose dimensions match the file record size (see 4.5.1.1).
5.1.3.2 Storage and Segmentation Declarations	See 5.1.2.2 and 4.5.4.	There are two memory optimization attributes: DENSE and ALIGNED. DENSE means that the amount of memory space occupied by the variable is more important than the time required to access it. Consequently the compiler will attempt to conserve storage space by packing items. The result of packing by the compiler is dependent on the target computer characteristics and the compiler implementation. ALIGNED means that the time required access this data is more important than the space it occupies. This attribute will cause the compiler to store the data for efficient access.

Non-executable statements

5.0	PL/1	HAL
5.1.3.3 Hardware Declarations	No	No
5.1.3.4 Format Declarations	The format declarations are written as part of the I/O activation statement. All conversions for variable types are provided. Much like Fortran - field widths, field repetition, positioning of characters, array and structure input by use of name without subscripts - repetition.	No

Non-Executable Statements

5.0	PL/1	HAL
5.2 Compiler Directives		
5.2.1 Optimization	The ORDER option specifies that the normal language rules are not to be relaxed; i.e., any optimization must be such that the execution of a block always produces a result that is in accordance with the strict definition of PL/1. This means that the values of variables set by execution of all statements prior to computational or system action interrupts are guaranteed in an on-unit entered as a result of the interrupt, or anywhere in the program afterwards. The REORDER option specifies that execution of a block must produce a result that is in accordance with the strict definition of PL/1 unless a computational or system action interrupt occurs during execution of the block; the result is then allowed to deviate as follows: 1. After a computational or system action interrupt has occurred during execution of the block, the values of variables modified, allocated, or freed in the block are guaranteed only after normal return from an on-unti or when accessed by the ONCHAR and ONSOURCE built-in functions.	In general optimization is not user-controlled with an exception below: 5.2.1.1 Space The DENSE attribute in a DECLARE statement directs the compiler to conserve storage space rather than access time. 5.2.1.2 Time The ALIGNED attribute in a DECLARE statement directs the compiler to conserve access time rather than storage space.

5.0	PL/1	HAL
	2. The values of variables modified, allocated, or freed in an on-unit for a computational or system action condition (or in a block activated by such an on-unit) are not guaranteed on return from the on-unit into the block, except for values modified by the ONCHAR and ONSOURCE pseudo-variables. The OPT compiler option, specified in the PARM field of the EXEC statement for a compilation, allows the programmer, to control the optimization for a particular compilation. For the fifth version of the F Compiler, the option can be specified with one of three values: OPT=0 requests fast compilation and, as a secondary consideration, reduction of the storage space required by the object program at the expense of execution time. OPT=1 requests fast compilation, and, as a secondary consideration, reduction of object program execution time at the expense of storage space. OPT=2 requests reduction of object program execution time at the expense of compilation time. The extra optimization phases of the compiler (i.e., those concerned mainly with loop and subscript optimization) are invoked only when OPT=2 is specified.	

Non-Executable Statements

5.0	PL/1	HAL
5.2.2 Debugging Aids	ON condition SNAP gives trace of calls when the given condition arises. Conditions CHECK and SUBSCRIPTRANGE are useful. See 4.3.5.	The ON statement helps with run-time debugging (see 4.3.5).
5.2.3 Documentation	Implementation Specific.	Implementation Specific.
5.2.4 Documentation Control	Implementation Specific.	Implementation Specific.
5.2.5 Storage Control	No	See 5.2.1.
5.2.6 Compilation Control	See 6.5.1.	See 6.5.1.
5.2.7 Others	No provision for inclusion of direct machine code.	No

Structure of Language and Compiler Interaction

6.0	PL/1	HAL
6.1 Self-Modification of Programs	No, however compile time facility provides this capability.	No, the REPLACE statement provides limited compile time facility (see 2.1.1.2).
6.2 Bootstrapping	No, but definitely possible. The pointer and based variables are provided to help with the problem.	Yes
6.3 Extensible	Yes, moderately; cannot extend data types.	HAL has a very limited extension facility in the REPLACE statement which can be used to create macros written in HAL. Data types cannot be modified.
6.4 Subsets or Dialects	Yes, the MULTICS PL/1 subset is compatible for the most part. Many dialects have been spawned.	There is a fictitious subset of HAL for scientific and engineering applications which has been described for users, but at the present time, a real subset of the compiler does not exist.
6.5 Debugging, Parametric Programming Facilities		
6.5.1 Compilation Time	Yes, the compile time facility which allows statements (preceded by % sign) identical to PL/1 statements for the editing of the source program run to compilation.	The REPLACE statement is a means of parameterization (see 2.1.1.2).

Structure of Language and Compiler Interaction

6.0	PL/1	HAL
6.5.2 Object Time	Yes, debugging and error checking with ON statement (see 4.3.5).	Yes, debugging and error checking with the ON and SEND statements (see 4.3.5).
6.6 Effect of Language Design on Implementation Efficiency, Remarks	PL/1 provides many more facilities for producing object-time efficiency than for improving compilation. Many of the features in the language have been chosen to aid the user at the expense of compilation time. There are a number of features in PL/1 which are there specifically to permit the compiler to generate optimized code, e.g., the ABNORMAL, NORMAL, IRREDUCIBLE, REDUCIBLE, and USES and SETS attributes. The RECURSIVE attribute avoids the need to make all object-time procedures recursive. PL/1 provides great flexibility to the user and, in particular, provides generality rather than restrictions. Among the specific features that have significant (but harmful) effect on compilation efficiency are the ability to use the key words as identifiers, the freedom state many options in any order that is desired, and the ability to factor attributes. The general problem of storage allocation has been given considerable attention at the language level in PL/1. A large amount of information is given to the compiler either directly or indirectly to assist it in doing reasonably good storage allocation.	The ability to partition arrays, matrices, vectors, hierarchical structures, and combinations of these with the use of the TO and AT operators in subscripting (see 1.3.5.1) provides unique versatility of arrayed arithmetic and decreases compiler efficiency. The conversion functions (INTEGER, SCALAR, BIT, CHARACTER, VECTOR, and MATRIX) when subscripted permit the creation of these data types and arrays of these types with the dimensions specified in the subscripts. This data-manipulation ability also adds inefficiency.

BUILT-IN FUNCTIONS FOR PL/1

1.0 Computational

A. String Handling

Function Name	Function Value
1. BIT (x,y)	x converted to a bit string of size y.
2. BOOL (x,y,z)	Produces a bit string (z) whose bit representation is a result of a given boolean operation on two given bit strings (x,y).
3. CHAR (x,y)	x converted to a character string of size y.
4. HIGH (x)	Character string of length x, composed of the highest characters in the collating sequence.
5. INDEX (x,y)	Searches a specified string (x) for a specified bit or character string configuration (y). If the configuration is found, the starting location of that configuration within the string is returned.
6. LENGTH (x)	Finds the string length of a given value and returns it to the point of invocation.
7. LOW (x)	Character string length x, composed of the lowest characters in the collating sequence.
8. REPEAT (x,y)	String x repeated y times.
9. STRING (x)	Concatenation of all the elements in an aggregate variable (x) into a single string element.
10. SUBSTR (x,y,z)	Substring of string x, starting at position y with length of z.

Figure 1

11.	TRANSLATE (s,r,[,p])	Source string S is translated by r and p which represent the replacement and position strings.
12.	UNSPEC (x)	Internally coded representation of x.
13.	VERIFY (string-1, string-2)	String-1 and string-2 are examined to verify that each character or bit in the first string is represented in second string.

B. Arithmetic

	Function Name	Function Value
1.	ABS (s)	Absolute value of x.
2.	ADD (w,x,y[,z])	w added to x; y and z are decimal integer constants that specify the precision of the result.
3.	BINARY (x,[,y[,z]])	x converted to binary base; y and z are decimal integer constants that specify the precision of the result.
4.	CEIL (x)	Smallest integer this is $\geq$ to x.
5.	COMPLEX (x,y)	Complex number with x as the real part and y as the imaginary part.
6.	CONJG (x)	Conjugate of x.
7.	DECIMAL (x,[,y[,z]])	x converted to decimal base; y and z are decimal integer constants that specify the precision of the result.
8.	DIVIDE (w,x,y[,z])	w divided by x; y and z are decimal integer constants that specify the precision of the result.

C. Mathematical

	Function Name	Function Value
1.	*ATAN (X[,z])	Arctan (x). z defines x/z.
2.	ATANH (x)	Arctanh (x).
3.	*COS (x,z)	Cos (x).
4.	COSH (x,z)	Cosh (x).
5.	ERF (x)	$(2/\sqrt{\pi})\ {}_0\int^x e^{-t2}dt$.
6.	ERFC (x)	1 - ERF (x).
7.	EXP (x)	e^x.
8.	LOG (x)	Log (x).
9.	LOG10 (x)	Log_{10} (x).
10.	LOG2 (x)	Log_2 (x).
11.	*SIN (x,z)	Sin (x).
12.	SINH (x,z)	$\sqrt{(x)}$
14.	*TAN (x)	Tan (x).
15.	TANH (x)	Tanh (x).

* A separate function is available for operands expressed in terms of degrees, rather than radians; e.g., ATAND (X), COSD, SIND, TAND.

9.	FIXED (x[,y[,z]])	x converted to fixed-point scale with y and z decimal integer constants specifying the precision of the result.
10.	FLOAT (x[,y])	x converted to a floating-point scale with a decimal integer constant specifying precision of result.
11.	FLOOR (x)	Largest integer not exceeding x.
12.	IMAG (x)	Imaginary part of complex number x.
13.	MAX (x,y,...)	Value of maximum argument.
14.	MIN (x,y,...)	Value of minimum argument.
15.	MOD (x,y)	Extracts the remainder resulting from the division of one real quantity by another and returns it to the point of invocation.
16.	MULTIPLY (w,x,y[,z])	w multiplied by x; y and z are decimal integer constants that specify the precision of the result.
17.	PRECISION (x,y[,z])	x converted to precision specified by decimal integers y and z.
18.	REAL (x)	Real part of complex number x.
19.	ROUND (Expression, n)	A fixed-point operand rounded at a specific point specified by n; or a floating-point operand with and bias removed.
20.	SIGN (x)	Returns real fixed binary value of 1 if x > 0, 0 if x = 0, and -1 if x < 0.
21.	TRUNC (x)	FLOOR(x), if x $\geq$ 0; otherwise CEIL(x), if x < 0.

D. Array Manipulation

Function Name	Function Value
1. ALL (X)	A bit string of the maximum length of any element of X, with a 1 wherever all the elements of X are 1; otherwise result is zero.
2. ANY (X)	A bit string of the maximum length of any element of X, with a 1 wherever any of the elements of X are 1; otherwise result is zero.
3. DIM (X,S)	Current extent of the Sth dimension of X.
4. HBOUND (X,S)	Current higher bound of the Sth dimension of X.
5. LBOUND (X,S)	Current lower bound of the Sth dimension of X.
6. POLY (a,x)	a (m:n) and x (p:q) are vectors with specified bounds and the result is: $a(m) + \sum_{j=1}^{n-m} (a(m+j) * \prod_{i=0}^{j-1} x(p+i))$
7. PROD (X)	The product of all the elements of X.
8. SUM (X)	The sum of all the elements of X.

2.0 Conditional

Function Name	Function Value
1. DATAFIELD	Varying length character string value of data field causing last NAME condition to be raised.
2. ONCHAR	Character which caused an I/O CONVERSION condition to be raised.
3. ONCODE	Code character identifying type of interrupt that caused entry into currently active on-unit.
4. ONCOUNT	Determines number of interrupts yet to be handled during abnormal completion of I/O event.
5. ONFILE	Determines file name on which the last CONVERSION or I/O operation was performed.
6. ONKEY	Value of key for record causing I/O condition to be raised.
7. ONLOC	Provides procedure name (entry point) in which ON-CONDITION interrupt occurred.
8. ONSOURCE	Contents of the last field being processed when the last CONVERSION interrupt occurred.

3.0 Based Storage (List Processing)

Function Name	Function Value
1. ADDR (X)	Finds the location (x) at which a given variable has been allocated and returns a pointer value to the point of invocation.
2. EMPTY	Clears an area of storage defined by an area variable.
3. NULL	Null point value; hence, it does not identify any generation of data.
4. NULLO	Returns NULL offset value so as to indicate that an offset variable does not currently identify an allocation.

4.0 Multitasking

Function Name	Function Value
1. COMPLETION (event name)	Determines the completion value of a given event variable.
2. PRIORITY (task name)	Priority of the named task relative to the priority of the task in which the function is evaluated.
3. STATUS (event name)	Determines the status value of a given event variable.

5.0 Miscellaneous

Function Name	Function Value
1. ALLOCATION (X)	"1" if storage has been allocated for major structure X; otherwise "0".
2. COUNT (filename)	Determines number of data items transmitted during the last GET or PUT operation.
3. DATE	Current date, in form YYMMDD, using year (Y), month (M), and day (D).
4. LINENO (filename)	Current line number of specified PRINT file.
5. TIME	Current time, in form HHMMSSTTT, using hours (H), minutes (M), seconds (S), and milliseconds (T).

BUILT-IN FUNCTIONS FOR HAL

1.0 Computational

A. String Handling

1. INDEX (string, config)

Arguments: Bit or character string. Searches a string for a specified bit or character configuration. The starting location of that configuration within the string is returned as an integer data type.

2. LENGTH (string)

Arguments: Bit or character string. Finds the string length and returns it as an integer data type.

3. LJUST (character-string)

Result: LJUST removes all the leading blanks of a character string operand and returns the resultant character string.

4. RJUST (character-string, p)

Result: RJUST creates a new character string of length, p. The character string argument is truncated on the left, or padded with blanks on the left, depending on whether its length is greater or less than p. p is a scalar expression which is rounded to the nearest integer before use.

B. Arithmetic Functions

Arguments: Bit strings, integers and scalars. These functions return the same data type as the argument (bit arguments are first converted to integers; the function returns an integer). Array arguments yield array results.

1. ABS

Finds the absolute value of the argument.

2. CEILING

   Determines the smallest integral value that is greater than or equal to the argument.

3. FLOOR

   Determines the largest integral value that does not exceed the argument.

4. ROUND

   Rounds the argument to nearest integral value.

5. SIGNUM

   Returns +1, 0, -1 as argument is positive, zero, and negative, respectively.

6. SIGN

   Returns +1, -1 as argument is positive or zero, and negative, respectively.

7. TRUNCATE

   Returns 0 if argument is less than +1 but greater than -1; otherwise equivalent of SIGN (argument) times the largest positive integral value that does not exceed ABS (argument).

8. MOD(a,b)

   MOD extracts the remainder c such that (a-c)/b=N where N is an integral number. c is the smallest positive number that must be subtracted from a in order to make N an integral number.

C. Mathematical

These functions return a scalar data type. Arguments may be bit, integer, or scalar. (Bits and integers are converted to scalars.) Array arguments yield array results.

1. ARCCOS

   Inverse trigonometric cosine; argument in closed interval [-1,1]; results in closed interval [0, π].

2. ARCCOSH

   Inverse hyperbolic cosine; arg not less than 1.

3. ARCSIN

   Inverse trigonometric sine; arg in closed interval [-1,1]; result in closed interval $[-\pi/2, \pi/2]$.

4. ARCSINH

   Inverse hyperbolic arc sine; arg any value.

5. ARCTAN

   Inverse trigonometric tangent; arg any value; result in open interval $[-\pi/2, \pi/2]$.

6. ARCTANH

   Inverse hyperbolic tangent; $|arg|<1$.

7. COS

   Trigonometric cosine; arg in radians; $|arg|<K1$.

8. COSH

   Hyperbolic cosine; $|arg|<K3$.

9. EXP

   Exponential, (e^{arg}); $|arg|<K3$.

10. LOG

    Natural logarithm; arg positive and non-zero.

11. SIN

    Trigonometric sine; arg in radians; $|arg|<K1$.

12. SINH

    Hyperbolic sine; $|arg|<K3$.

13. TAN

Trigonometric tangent; arg in radians; arg not odd multiple of $\pi/2$; $|arg|<K2$.

14. TANH

Hyperbolic tangent; arg any value.

15. SQRT

Square root; arg positive.

Note: K1, K2 and K3 are upper limits which depend upon machine characteristics.

D. Array

These functions have the following general format:

function-label(single operand)

where the function will operate on the "linear array" representing the "inner-most" free index of the argument. The single-operand may be of bit, integer, scalar, vector, or matrix data types or arrays of these types. The following table indicates the array shape and dimension of the function result.

Argument	$[X]_{a}^{(1)}$	$[X]_{a,b}^{(1)}$	$\overline{V}_{\ell}$	$[\overline{V}]_{a,b:\ell}$	$\overset{*}{M}_{m,n}$	$\overset{*}{M}_{a,b:m,n}$
Result	$A^{(2)}$	$[A]_{a}^{(2)}$	$S^{(3)}$	$[S]_{a,b}^{(3)}$	$\overline{V}_{m}$	$[\overline{V}]_{a,b:m}$

Subscripts indicate shape and dimension (i.e., <array-shape>:<dimensions>) $\ell \equiv$ vector length; m,n $\equiv$ matrix rows, columns; a,b $\equiv$ array shape. (In general, the argument array shape may be a,b,c,...etc.)

NOTES:

(1) X may be bit string, integer or scalar.

(2) A is an integer if X is a bit string or integer.

(3) S indicates scalar.

(4) [ ] indicates array.

The linear array functions are:

1. SUM

   Sums over inner-most free-index.

2. PROD

   Forms product over inner-most free index.

3. MAX

   Finds maximum element value over inner-most free index.

4. MIN

   Finds minimum element value over inner-most free index.

NOTE: "free-index" means subscript level with a free range of values rather than a single index value assigned. The functions operate only over one dimension (one subscript range).

E. Matrix-Vector

Arguments may be vectors or matrices (as applicable). Array arguments yield array results.

1. ABVAL

   Absolute value of magnitude of vector; argument may be a vector of any length.

2. ADJ

   Adjoint; argument is invertible square matrix of any dimension; result is equal to

   DET(argument)times INVERSE(argument).

3. DET

   Determinant; argument is a square matrix.

4. INVERSE

   Inverse; argument is square matrix; result is inverse if argument is invertible.

5. TRACE

Trace; argument is square matrix; result is sum of diagonal matrix elements.

6. TRANSPOSE

Transpose, argument is matrix of any dimensions; result is the interchange of the rows and columns of the argument.

7. UNIT

Unit vector; argument is vector of any length; result is a vector of magnitude 1 and in line with argument.

F. Data-Type Conversion and Shaping

Arguments: Bit, integer, scalar, vector, matrix, character, and arrays of these types. Arrayed arguments yeild arrayed results. Matrix and vector arguments with INTEGER, SCALAR, BIT, and CHARACTER functions also yield array results.

1. INTEGER

Converts argument to integer

2. SCALAR

Converts argument to scalar.

3. BIT

Converts argument to bit string.

4. CHARACTER

Converts argument to character.

5. VECTOR

Converts argument to vector.

6. MATRIX

Converts argument to matrix.

These functions when subscripted can be used to create data types (same as function names) of the dimensions specified in the subscripts. For example:

$$MATRIX_{3,5:3,3}(argument\text{-}list)$$

would create a 3x5 array of 3x3 matrices. The argument list may contain data types and mixtures of types listed above (under "Arguments"). Repetition may be indicated with the # operator preceded by the number of repetitions, and followed by an argument or parenthesized list to be repeated.

2.0 Conditional

None

3.0 Based Storage (List Processing)

None

4.0 Multitasking

None

5.0 Miscellaneous

1. RANDOM

   Result is the current base random number in the pseudo-random number generator. This function enables the programmer to make successive runs of a program without repeating sequences of pseudo-random numbers.

2. RANDOMG

   Selects a random number from a Gaussian distribution.

3. TIME

   Returns current time as integer.

4. DATE

   Returns current data as integer.

Pseudo-variables

A pseudo-variable, in HAL, is a function that can only appear on the left of an equal sign (=) in an assignment or DO statement. The only defined pseudo-variable is SUBBIT. SUBBIT permits bit strings to be assigned directly to the bit representation of a variable. Argument is the variable name.

GROUP 2 LANGUAGES

INTRODUCTION

The JOVIAL language is a multi-purpose language that is related to the International Algebraic Language or later, ALGOL 58. JOVIAL was designed to be used by professional programmers for solving large scale information system problems. It has scientific/numeric capabilities as well as character and bit string handling capabilities. JOVIAL is principally used by military organizations for solving command and control problems. It generally exists as Basic JOVIAL and JOVIAL (J(3)) where the latter is an extension of the former. However, there are many other dialects of JOVIAL (J(4)), that exist. The digit following the "J" in the code name for a version of JOVIAL indicates the sophistication of that language. Theoretically, the features of any version of JOVIAL (J(N)) form a subset of its successor language (J(N+1)). Thus, one can see why many JOVIAL compilers were written in JOVIAL.

The SPL/J6 language is a multi-purpose language that is a dialect of JOVIAL (J(6)) with many extensions and a different notation. The language was designed for space software applications. SPL/J6 has the capabilities of JOVIAL plus features allowing decision tables, real time control and non-scalar algebraic operations. This specification is used in the comparison tables. However, there are five dialects of the SPL/J6 language: SPL/MK I to SPL/MK V. The MK I subset is designated as CLASP. A compiler for CLASP was operational in late 1970. MK II is defined for a fixed point aerospace computer and MK III for a floating point aerospace computer. MK IV is defined for a ground based support computer and MK V for a ground based multiprocessor. A MK II compiler is available and in use on the SPARS program.

1.0	Form of Language		
	JOVIAL (J3)	SPL/J6	CLASP
1.1. Character Set			
1.1.1 Upper Case Letters	26 letters of English alphabet - A to Z	26 letters of English alphabet - A to Z	26 letters of English alphabet - A to Z
1.1.2 Lower Case Letters	No	No	No
1.1.3 Arabic Numerals	10 Arabic numerals - 0 to 9	10 Arabic numerals - 0 to 9	10 Arabic numerals - 0 to 9
1.1.4 Special Marks or Signs	12 special marks as follows: + (plus sign) - (minus sign) * (asterisk) / (slash) b (blank, space) . (period, decimal point) , (comma) = (equal sign) ((left parenthesis)) (right parenthesis) ' (prime) $ (dollar sign) Some implementations include EBCDIC codes with the Hollerith codes.	12 special marks as follows: + (plus sign) - (minus sign) * (asterisk) / (slash) b (blank, space) . (period, decimal point) , (comma) = (equal sign) ((left parenthesis)) (right parenthesis) ' (prime) $ (dollar sign) EBCDIC codes may be included with Hollerith codes.	12 special marks as follows: + (plus sign) - (minus sign) * (asterisk) / (slash) b (blank, space) . (period, decimal point) , (comma) = (equal sign) ((left parenthesis)) (right parenthesis) ' (prime) $ (dollar sign)

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.2 Types of Basic Elements			
1.2.1 Language Defined			
1.2.1.1 Keywords or Primitives	A ABS ALL AND ARRAY ASSIGN* B BEGIN BIT BYTE CHAR CLOSE D DEFINE DIRECT E END ENT ENTRY EQ EQUATE EXECUT F FILE FOR GOTO GQ GR H I IF IFEITH INPUT ITEM JOVIAL* L LOC LQ LS M MANT MODE N NENT NOT NQ NWDSEN O ODD OPEN OR ORIF OUTPUT OVERLAY P POS PROC R REM REMQUO RETURN S SHUT START STOP STRING SWITCH T TABLE TERM TEST U V *Appears only in direct code.	A ABS ACTIONS AND ARRAY AS B BACKSPACE BIN BOOLEAN CLOSE CONDITIONS CONSTANT COUNT DECLARE DEFINE DO EJECT ELSE END ENDALL ENTRANCE EQ EQUIV EXIT F FALSE FILE FIXED FOR FLOATING GOTO GQ GR HARDWARE HEX I IF INDEX INLINE INTEGER INTERRUPT ITEM LAND LOC LOCAL LOCK LQ LS LSH LXOR MODE NENT NOT NQ O OCT OFF OPEN OPTIMIZE OR ORIF OVERLAY PATTERN PRESET PROC READ RECURSIVE REENTRANT REM REMQUO REWIND S SEARCH SHUT SKIP STATUS STOP STORAGE SWITCH TABLE TEXT THEN TIME TRACE TRUE UNLOCK UNTIL UNTRACE WAIT WHILE WRITE X .S RETURN RSH	A ABS AND B BOOLEAN BY CLOSE CONSTANT COUNT DECLARE ELSE END ENDALL EQ EQUIV EXIT F FALSE FIXED FLOATING FOR GOTO GQ GR I IF INTEGER L LAND LIM LOCK LOR LQ LS LSH LXOR NOT NQ O OFF ON OR OVERLAY PACK PROC REMQUO RND RSH SIC SIGN START STOP T TERM TEXT THEN TO TRACE TRUE UNCOUNT UNLOCK UNPACK UNPACX UNSIC UNSPACE UNTRACE UNTIME X .S

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.2.1.2 Operators			
1.2.1.2.1 Physical Graphics	+ - * / ** or (* *)	+ - * / ** ABS '	+ - * / ** /*/
1.2.1.2.2 Keywords	LS LQ EQ GR GQ NQ AND OR NOT GOTO IF IFEITH ORIF FOR TEST CLOSE RETURN STOP INPUT OUTPUT OPEN SHUT	LS LQ EQ GR GQ NQ AND OR EQUIV NOT LAND LXOR LOR RSH LSH .S	LS LQ EQ GR GQ NQ AND OR EQUIV NOT LAND LXOR LOR RSH LSH .S

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.2.1.3 Punctuation			
1.2.1.3.1 Brackets			
1.2.1.3.1.1 Physical Graphics	( ) ($ $) (* *) (/ /) " "	( ) " " ' '	( ) " " ' '
1.2.1.3.1.2 Keywords	START TERM BEGIN END DIRECT JOVIAL	START TERM (NO "BEGIN") END ENDALL DIRECT END	START TERM (NO "BEGIN") END ENDALL DIRECT END
1.2.1.3.2 Separators			
1.2.1.3.2.1 Physical Graphics	. , = == ' $	. , = ==	. , = == $

1.0	Form of Language JOVIAL (J3)	SPL/J6	CLASP
1.2.1.3.2.2 Keywords		BY UNTIL WHILE AS	BY TO
1.2.2 Programmer Defined			
1.2.2.1 Identifiers	Yes	Yes	Yes
1.2.2.2 Constants (Literals)			
1.2.2.2.1 Numeric	Fixed Point Floating Point Integer Decimal Dual (Fixed Point only) Octal	Fixed Point Floating Point Integer Binary Decimal Hexadecimal Octal	Fixed Point Floating Point Integer Binary Decimal Hexadecimal Octal
1.2.2.2.2 Textual	Yes	Yes	Yes
1.2.2.2.3 Boolean	Yes, plus status constant	Yes, plus status constant	Yes
1.2.2.2.4 Other	Location	Location	Location

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.2.2.3 Comments	Comments can be embedded in any statement.	Comments can be embedded in any statement.	Comments can be embedded in any statement.
1.3 Identifier Definition			
1.3.1 Types of Identifiers			
1.3.1.1 Statement Names	Yes	Yes	Yes
1.3.1.2 Data Names	Yes	Yes	Yes
1.3.1.3 Index Variable Names	Yes	Yes	Yes
1.3.1.4 Label Variable Names	No	No	No
1.3.1.5 Others	Procedure names	Procedure names	Procedure names

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.3.2 Formation Rules for Identifiers	An identifier must start with a letter, followed by at least one letter or numeral, which may be punctuated for readability by the ' separator. They cannot end with prime nor contain two consecutive primes. A period must be used to separate the identifier from the statement body. Index variables are designated in FOR statements and are designated by a single letter. Identifiers cannot be longer than 120 characters.	An identifier must start with a letter followed by up to 120 letters or numerals. A period must be used to separate the identifier from the statement body.	An identifier must start with a letter which may be followed by from 0 to 7 letters or digits. A period must be used to separate the identifier from the statement body.
1.3.3 Use of Reserved Words	Identifiers, in general, cannot be the same as keywords. However, there are several keywords that are not reserved.	Identifiers, in general, can be the same as keywords except where their use results in an ambiguous syntax.	Identifiers, in general, cannot be the same as keywords. However, there are several keywords that are not reserved.
1.3.4 Synonyms			
1.3.4.1 Preset	Two or more names can be assigned to occupy the same storage area with EQUATE statement and variation of OVERLAY declaration.	Variation of OVERLAY declaration.	Variation of OVERLAY declaration.
1.3.4.2 Dynamic	No	No	No

	Form of Language		
1.0	JOVIAL (J3)	SPL/J6	CLASP
1.3.5 Structure of Data Names			
1.3.5.1 Subscription	References to TABLE, TABLE-STRING and ARRAY declarations are followed by subscript expressions separated by commas and enclosed in ($ $) pair. The number of subscript expressions must equal the number of dimensions of aggregate. NENT, NWDSEN, ALL, ENTRY functional modifiers are used with TABLE processing and have attributes of subscription. Array subscripts are ordered by rows, columns, planes, etc.	References to TABLE and ARRAY declarations are followed by subscript expressions separated by commas and enclosed in ( ) pair. The number of subscript expressions must equal the number of dimensions of aggregate. Subscripting can be used to reference many elements (vectors in matrices). Subcripts themselves may be subscripted. Array names do not have to be repeated if common to all subscripts; array name is written once, followed by list of subscripts. Implied subscripts may be explicitly declared with an array or table. Partial indexing allows references to bits or bytes (scalars) or elements (non-scalar) of arrays or tables (/./). NENT can be used as the subscript to obtain the number of entries in tables or arrays. Array subscripts are ordered by rows, columns, planes, etc.	References to declared arrays are followed by subscript expressions separated by commas and enclosed in ( ) pair. The number of integer expressions must equal the number of dimensions with which the array was declared. Subscript notation has been extended to include the concept of cross section. Array subsubscripts are ordered by rows, columns and planes

1.0	Form of Language JOVIAL (J3)	SPL/J6	CLASP
1.3.5.1.1 Dimensions	1-N dimensions	1-N dimensions	1-3 dimensions
1.3.5.1.2 Classes	Constants, variables and any arithmetic expression that can be evaluated and converted to integer form may be used as a subscript which includes other subscripted variables. This includes switch indices.	Constants, variables and any arithmetic expression that can be evaluated and converted to integer form may be used as a subscript. Subscripted expressions can occur within subscripted expressions to any level. This includes switch indices. Cross section is specified by the absence of that particular subscript. Implied subscripts in array declarations.	Restricted to simple integer formula of variable multiplied or divided by a constant plus or minus some other constant. The * used in subscript expression to mean cross section of array.
1.3.5.1.3 Form	Fractional bits are truncated to give integer.	Fractional bits are truncated to give integer.	No
1.3.5.1.4 Data Allowable Elements	Any data types can occur in TABLE, TABLE-STRING and ARRAY declarations. JOVIAL tables are useful for more efficient storage allocation. There is no hierarchy associated with JOVIAL aggregates.	Any data types can occur in TABLE and ARRAY declarations. SPL tables are useful for more efficient storage allocation. SPL does allow hierarchical groupings of data.	Integer constants can occur in array declarations. CLASP does allow limited hierarchical groupings of data.

Form of Language.

1.0	JOVIAL (J3)	SPL/J6	CLASP
1.3.5.2 Qualification	No	The group identifier allows entire data declarations to be assigned a group name. This group name allows identical identifiers to be used for items, tables and arrays if they appear in different groups. Qualified names are specified left to right and separated by the single prime. The maximum depth of qualified names is three - group, table, item.	A group is declared by labeling a data declaration statement. The label must be followed by a period. Used to name groups of single data items such that a group as a whole can be referenced by group name. Elements can have different attributes.
1.3.5.3 Qualification and Subscription	No	Indexing follows the same rules as ordinary indexing.	No
1.3.5.4 Numbering Conventions	All elements of aggregates are numbered consecutively from zero to number of elements minus one.	All elements of aggregates are numbered consecutively from zero to number of elements minus one. See 5.2.7 NUMBER ORIGIN.	All elements of aggregates are numbered consecutively from zero to number of elements minus one.

.0	Form of Language		
	JOVIAL (J3)	SPL/J6	CLASP
.3.6 Scope of Names Based on or Relative to Program Structure			
.3.6.1 System Names (Shared)	Names defined in the Compool are defined for an entire program system, excluding programs or procedures that define identically spelled names.	Names defined in the Compool are defined for an entire program system, excluding programs or procedures that define identically spelled names.	No
.3.6.2 Global Names (Shared)	See figure (1)	Statement labels can be referenced anywhere within the program and have the program as their total scope. Data names with certain exceptions, can be referenced throughout the program.	Not explicitly discussed in manual.
.3.6.3 Local Names	See figure (1). Names defined within a procedure override names defined elsewhere.	Data names may be locally declared through a special declaration (LOCAL) and data names declared within a procedure are local to that procedure. Names defined within a procedure override names defined elsewhere.	Not explicitly discussed in manual.

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.4 Definition and Usage of Other Basic Elements			
1.4.1 Operators			
1.4.1.1 Arithmetic Operators			
1.4.1.1.1 Scalar	+ (Addition) - (Subtraction) * (Multiplication) / (Division) ** or (* *) (Exponentiation)	+ (Addition) - (Subtraction) * (Multiplication) / (Division) ** (Exponentiation) ABS (Absolute Value)	+ (Addition) - (Subtraction) * (Multiplication) / (Division) ** (Exponentiation)
1.4.1.1.2 Non-Scalar	No	' Matrix Transpose **-1 Matrix Inverse + Matrix-Scalar Sum - Matrix-Scalar Difference * Matrix-Scalar or Scalar Matrix or Matrix-Matrix Product Vectors are defined as any one dimensional array where a row vector is 1xN and a column vector is Nx1. Operations such as vector inner and outer product can be done but they must use the transpose operation.	+ Matrix-Scalar Sum - Matrix-Scalar Difference * Matrix-Scalar or Scalar-Matrix Product |*| Matrix-Matrix Product / Matrix-Scalar or Matrix-Matrix Division Vectors are defined as in SPL. Asterisks allow reference to whole rows or columns or matrices. Certain vectors operations such as vector outer product and vector inner product can't be done because there is no transpose operation.

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.4.1.2 Comparison Operators			
1.4.1.2.1 Binary	LS (less than) LQ (less than or equal) EQ (equal) GR (greater than) GQ (greater than or equal) NQ (not equal)	LS (less than) LQ (less than or equal) EQ (equal) GR (greater than) GQ (greater than or equal) NQ (not equal)	LS (less than) LQ (less than or equal) EQ (equal) GR (greater than) GQ (greater than or equal) NQ (not equal)
1.4.1.2.2 Unary	By use of constant zero (0) for right part of expression.	By use of constant zero (0) for right part of expression.	By use of constant zero (0) for right part of expression.
1.4.1.3 Boolean Operators			
1.4.1.3.1 Binary	AND (logical conjunction) OR (logical disjunction)	AND (logical conjunction) OR (logical disjunction) EQUIV (logical equivalence)	AND (logical conjunction) OR (logical disjunction) EQUIV (logical equivalence)
1.4.1.3.2 Unary	NOT (logical negation)	NOT (logical negation)	NOT (logical negation)
1.4.1.4 Bit String Operators			
1.4.1.4.1 Binary	No	LAND (logical product) LOR (logical sum) LXOR (exclusive or)	LAND (logical product) LOR (logical sum) LXOR (exclusive or)

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.4.1.4.2 Unary	No	RSH (right shift) LSH (left shift)	RSH (right shift) LSH (left shift)
1.4.1.5 Functional Modifiers	BIT BYTE CHAR MANT ODD — bit,string processing ALL ENTRY (ENT) NENT NWDSEN POS LOC — table processing	LOC NENT	LOC
1.4.1.6 Others	No	.S (Scaling operator)	.S (Scaling operator)
1.4.2 Delimiters	Operators and delimiters have same function.	Operators and delimiters have same function.	Operators and delimiters have same function.

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.4.2.1 Brackets	( ) (expression grouping, parameter lists, Hollerith codes, certain item or string names) ($ $) (subscripts) (* *) (exponents) (/ /) (absolute value) " " (comments) START TERM (program unit) BEGIN END (blocking) DIRECT JOVIAL (machine code)	( ) (expression grouping, parameter lists, subscripts) " " (comments) ' ' (textual data) START TERM (program unit) (No "begin") END ENDALL (compound statements) DIRECT END (machine code)	( ) (expression grouping, parameter lists, subscripts) " " (comments) ' ' (textual data) START TERM (program unit) (No "begin") END ENDALL (compound statements) DIRECT END (machine code)
1.4.2.2 Separators	. (statement labels, indicates binary, octal, and decimal points) , (parameter list items) = (assignment) == (exchange) ' (identifiers for readability) $ (statement terminator) ƀ (adjacency, separates elements of statement) ASSIGN (machine register to item and vice versa)	. (statement labels, indicates binary, octal, decimal and hexadecimal points) , (parameter list items) = (assignment) == (exchange) ƀ (adjacency, separates elements of statement)	. (statement labels, indicates binary, octal, decimal and hexadecimal points) , (parameter list items) = (assignment) == (exchange) $ (statement terminator for statements on same line) ƀ (adjacency, separates elements of statement)

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.4.3 Punctuation	( ) pair precede and follow arithmetic operands, function-procedure parameters and determine priority of computation. If parentheses are nested, the innermost are evaluated first. ($ $) enclose subscript expressions. (* *) may enclose exponents (/ /) causes absolute value of enclosed expression. " " encloses comments. START TERM pair encloses the list of imperative or declarative statements. BEGIN END encloses compound statements. DIRECT JOVIAL encloses direct machine code. . used to identify statements by preceding them with a name and a period. , used to separate lists of items or parameters. = used to designate assignment statements. == used to designate exchange statement labels.	( ) pair precede and follow arithmetic operands, function-procedure parameters, subscripts, and determine priority of computation. If parentheses are nested the innermost are evaluated first. ' ' pair encloses textual data. " " encloses comments. START TERM pair encloses the list of imperative or declarative statements. (No "begin") END ENDALL Compound statements have implied "begin" and are enclosed in END or ENDALL. DIRECT END encloses direct machine code. . used to identify statements by preceding them with a name and a period. , used to separate lists of items or parameters. = used to designate assignment statements. == used to designate exchange statements.	( ) pair precede and follow arithmetic operands, procedure parameters, subscripts and determine priority of computation. If parentheses are nested the innermost are evaluated first. ' ' pair encloses textual data. " " encloses comments. START TERM pair encloses the list of imperative or declarative statements (No "begin") END ENDALL Compound statements have implied "begin" and are enclosed in END or ENDALL. DIRECT END encloses direct machine code. . used to identify statements by preceding them with a label and a period. , used to separate lists of items or parameters. = used to designate assignment statements.

1.0	Form of Language		
	JOVIAL (J3)	SPL/J6	CLASP
	' used to link for readability. $ is used to end declarative and imperative statements. ♭ Names and constants may not be immediately adjacent. They must be separated by either an operator, assignment mark, parenthesis, comma, period, blank or "comment" "comment". CLOSE END encloses closed subroutines. PROC END encloses procedures.	CLOSE EXIT encloses closed subroutine declaratives. PROC EXIT encloses procedure declarations. ♭ Names and constants may not be immediately adjacent. They must be separated by either an operator, assignment mark, parenthesis, comma, period, blank or "comment".	== used to designate exchange statements. $ is used to end imperativ named statements on the same line or card. Otherwise statements need not end with "$". ♭ Names and constants may not be immediately adjacent. They must be separated by either an operator, assignment mark, parenthesis, comma period, blank or "comment". CLOSE EXIT encloses closed subroutine declaratives. PROC EXIT encloses procedure declarations. ON EXIT encloses chronic statements.

1.0	Form of Language: JOVIAL(J3)	SPL/J6	CLASP
1.4.4 Significance of Blanks	Blanks are significant and must not be embedded in imperative or declarative names. Blanks must follow floating or fixed constants unless the first letter or numeral of the next term is a special mark other than $ mark. The last alphanumeric of a term and the first letter of the following must be separated by a space. Blanks cannot be embedded in keywords. Where blanks are allowed any number of blanks is acceptable.	Blanks are significant and must not be embedded in imperative or declarative labels. Blanks cannot be in the keywords. One or more spaces must separate SPL words. Separating spaces may be omitted if the omission does not join two letters, two numbers, or a letter and a number.	Blanks are significant and must not be embedded in imperative or declarative labels. Blanks cannot be embedded in the keywords. Where blanks are allowed any number of blanks is acceptable.
1.4.5 Literals	Numeric and Hollerith (EBCDIC) or standard transmission code. Hollerith literals are enclosed in parenthesis and preceded by decimal number of characters in string.	Numeric and Hollerith (EBCDIC). Hollerith literals are enclosed in single quotes and optionally preceded by decimal number of characters in string. This option is for occurrence of primes in string.	Numeric and Hollerith. Hollerith literals are enclosed in single quotes and optionally preceded by decimal number of characters in string. This option is for occurrence of primes in string.
1.4.6 Noise Words	No	No	No

1.0	Form of Language: JOVIAL (J3)	SPL/J6	CLASP
1.5 Input Format			
1.5.1 Physical Input Format			
1.5.1.1 Linear			
1.5.1.1.1 Fixed (columnar restrictions)	No	No	No
1.5.1.1.2 String	String oriented, with coding beginning in any column from 1 to 72. Card columns 73-80 are not recognized by compiler. The compiler assumes that card column 1 of the next card is to be processed next. The amount of continuation is also a function of the implementation. There can be more than one statement per line.	String oriented, with coding beginning in any column from 1 to 72. Card columns 73-80 are not recognized by compiler. The compiler assumes that card column 1 of the next card is to be processed next. The amount of continuation is a function of the implementation. There can be more than one statement per line.	String oriented, with coding beginning in any column from 1 to 72. Card columns 73-80 are not recognized by compiler. Card column 73 is used for continuation to next statement. Any non-blank character will do. There is no restriction on the amount of continuation. There can be more than one statement per line but they must be separated by ($) mark. If continuation field is blank compiler assumes next card is a new statement
1.5.1.1.3 Combination	No	No	No

	Form of Language		
1.0	JOVIAL(J3)	SPL/J6	CLASP
1.5.1.2 Non-Linear			
1.5.1.2.1 Fixed (columnar restrictions)	No	No	No
1.5.1.2.2 String	No, notation does not clearly resemble statement of problem or method of solution.	No, notation does not clearly resemble statement of problem or method of solution.	No, notation does not clearly resemble statement of problem or method of solution.
1.5.1.2.3 Combination	No	No	No
1.5.2 Conceptual Form			
1.5.2.1 Symbolic or Formal	No	No	No
1.5.2.2 English Like	No	No	No
1.5.2.3 Pseudo English Like	Language is more oriented to engineering/scientific notation. Reference language is hardware language.	Language is more oriented to engineering/scientific notation. Reference language is hardware language.	Language is more oriented to engineering/scientific notation. Reference language is hardware language.

Reproduced from best available copy.

Structure of Program

2.0	JOVIAL (J3)	SPL/J6	CLASP
2.1 Statement Types			
2.1.1 Non-Executable Statements			
2.1.1.1 Declaratives	ARRAY DEFINE FILE ITEM MODE OVERLAY PROC (definition) STRING SWITCH TABLE	ARRAY close definition DECLARE FILE HARDWARE INDEX ITEM LOCAL MODE OVERLAY PROC (definition) TABLE	close definition DECLARE OVERLAY PROC (definition)
2.1.1.2 Compiler Directives	DIRECT JOVIAL	COUNT END DEFINE END, AS END with AND, OR, PAT connectors PATTERN END DIRECT END NUMBER ORIGIN = OPTIMIZE SPACE END TIME END STORAGE = TRACE UNTRACE	COUNT UNCOUNT DIRECT END OPTIMIZE SPACE UNSPACE TIME UNTIME SIC UNSIC TRACE UNTRACE

Structure of Program

2.0	JOVIAL (J3)	SPL/J6	CLASP
.1.1.3 Comments	Any character string (letters, numerals or special marks) enclosed by double primes. It may contain spaces. It must not contain two double primes in succession. It must not contain $ except as ($, or $). Double primes must not occur within the enclosed string.	Any character string (letters, numerals or special marks) enclosed by double primes. It may contain spaces. It must not consist of two double primes in succession. Double primes must not occur within the enclosed string.	Any character string (letters, numerals or special marks) enclosed by double primes. It may contain spaces. It must not consist of two double primes in succession. Double primes must not occur within the enclosed string.
.1.2 Executable Statements			
.1.2.1 Smallest Executable Statement	Single statement of the form ABLE = 1$	Single statement (simple) of the form ABLE = 1	Single statement of the form ABLE = 1

Reproduced from best available copy.

Structure of Program

2.0	JOVIAL (J3)	SPL/J6	CLASP
2.1.2.2 Grouped Executable Statements			
2.1.2.2.1 Block Structure	Combined individual statements form compound statements (IF,FOR) by enclosing with BEGIN END brackets. A compound statement must contain at least one statement that is an executable statement. Compound statements may be nested to any level.	Compound conditional statements (IF, THEN, ELSE, ORIF) and loop statements (FOR) both of which can be nested. Decision tables (CONDITIONS ACTIONS).	Compound conditional statements (IF, THEN, ELSE and loop statements (FOR) both of which can be neste
2.1.2.2.2 Others	No	INLINE attribute of PROC for macro expansion.	INLINE attribute of PROC for macro expansion.
2.1.2.3 Loops	Loops are handled by FOR imperative. TEST imperative can be used for conditional exits. IF imperative can also be used.	Loops are handled by the FOR imperative. IF imperative can also be used.	Loops are handled by the FOR imperative. IF imperative can also be used.

Structure of Program

2.0	JOVIAL (J3)	SPL/J6	CLASP
2.1.2.4 Procedures, Coprocedures, Functions or Subroutines	Procedures, functions and closed subroutines are available through the PROC and close declaratives. Coprocedures are not part of language.	Procedures, functions and closed or open subroutines are available through the PROC, close, declaratives. Recursion (RECURSIVE) is allowed. Coprocedures are not part of language.	Procedures, functions and closed or open subroutines are available through the PROC, close declaratives. Coprocedures are not part of language.
2.1.2.5 Inclusion of Other Languages			
2.1.2.5.1 Assembly Language (AL)	Yes	Yes	Yes
2.1.2.5.2 Assembly Routines	Yes	Yes	Yes
2.1.2.5.3 Higher Order Language (HOL)	No	No	No
2.1.2.5.4 HOL-AL Communication	Through ASSIGN declarative	No	No

Structure of Program

2.0	JOVIAL (J3)	SPL/J6	CLASP
2.1.3 Intermingling Order for Non-Executable and Executable Statements	The START TERM brackets enclose all declarative and imperative statements. The START keyword can optionally contain the word CLOSE which declares program as a closed subroutine. TERM can optionally point to first statement to be executed. Declarative statements can be mixed with imperatives, but declaratives must be defined before they can be referred to in an imperative statement. CLOSE subroutines can be located anywhere among the imperative statements of a program. Switch declarations may a appear anywhere in program. A system may consist of separate main programs each of which can contain procedures.	The START TERM brackets enclose all declarative and imperative statements. Declarative statements can be mixed with imperatives but declaratives must be defined before they can be referred to in an imperative statement. CLOSE subroutines can be located anywhere among the imperative statements of a program. Switch declarations may appear anywhere in program. Subprograms are included within main programs.	The START TERM brackets enclose all declarative and imperative statements. Declarative statements can be mixed with imperatives but declaratives must be defined before they can be referred to in an imperative statement. CLOSE subroutines must be located within the loop statement group if they reference variables which are active outside the CLOSE. Subprograms are included within main programs.
2.1.4 Operating System Interface	Through OPEN, SHUT, INPUT, OUTPUT, POS imperatives and using FILE declarations to describe data.	Through OPEN, SHUT, READ, WRITE imperatives and using FILE declarations to describe data.	No

2.0	Structure of Program		
	JOVIAL (J3)	SPL/J6	CLASP
2.2 Statement Characteristics			
2.2.1 Methods of Delimiting			
2.2.1.1 Explicit	See 1.4.3. Main delimiter is dollar sign ($).	See 1.4.3.	See 1.4.3.
2.2.1.2 Implicit	Yes	Compound statements (IF, FOR) have implied "begin" and terminate with END. ENDALL used to eliminate need for string of ENDS in nested statements.	Compound statements (IF, FOR) have implied "begin" and terminate with END. ENDALL used to eliminate need for strings of ENDS in nested statements.
2.2.2 Parameter Passage Required (Different Data Types)			
2.2.2.1 Call by Name			
2.2.2.1.1 Formal			
2.2.2.1.1.1 Input	Array name, table name, Hollerith item name, close name	Array name, table name, literal item name, close name.	No

2.0	Structure of Program		
	JOVIAL (J3)	SPL/J6	CLASP
2.2.2.1.1.2 Output	Array name, table name, Hollerith item name, statement label	Array name, table name, literal item name, statement label	No
2.2.2.1.2 Calling			
2.2.2.1.2.1 Input	Array name, table name, literal item name, close name	Array name, table name, literal formula, close name	No
2.2.2.1.2.2 Output	Array name, table name, Hollerith formula, close name	Array name, table name, literal variable, statement label	No
2.2.2.2 Call by Value			
2.2.2.2.1 Formal			
2.2.2.2.1.1 Input	Item name	Simple item name	Non-textual item name.
2.2.2.2.1.2 Output	Item name	Simple item name	Non-textual item name.

Structure of Program

2.0	JOVIAL (J3)	SPL/J6	CLASP
2.2.2.2.2 Calling			
2.2.2.2.2.1 Input	Item formula	Simple item formula	Non-textual formula
2.2.2.2.2.2 Output	Item variable	Simple item variable	Non-textual variable
2.2.2.3 Call by Address			
2.2.2.3.1 Formal			
2.2.2.3.1.1 Input	No	No	Array name, textual item name, close name.
2.2.2.3.1.2 Output	No	No	Array name, textual item name, statement label.
2.2.2.3.2 Calling			
2.2.2.3.2.1 Input	No	No	Array name, textual formula, close name.
2.2.2.3.2.2 Output	No	No	Array name, textual variable, statement label.

Structure of Program

2.0	JOVIAL (J3)	SPL/J6	CLASP
2.2.3 Embedding	Allowed with use of the IF, ORIF, IFEITH statements to any depth. Arithmetic expressions can contain functions and Boolean expressions. Procedure calls can be embedded in statements. Procedures can invoke other procedures. Embedded comments act as a blank. Compound statements can be contained within compound statements.	Allowed with use of the IF-THEN-ELSE statement to any depth. Arithmetic expressions can contain functions, Boolean and logical expressions. In addition, ELSE and ORIF clauses can be used. Procedure calls can be embedded in statements. Procedures can invoke other procedures. Embedded comments act as a blank. Compound statements can be contained within compound statements.	Allowed with use of the IF-THEN-ELSE statement to any depth. In addition the ELSE clause can be used. Arithmetic expressions can contain functions, Boolean and logical expressions. Embedded comments act as a blank. Compound statements can be contained within compound statements.
2.2.4 Recursion	No	RECURSIVE attribute tagged to a PROC allows procedure to call self. It is not automatic and must be explicitly specified.	No
2.2.5 Reentrant			
2.2.5.1 Serially Reusable	Implementation specific	Implementation specific	Implementation specific
2.2.5.2 Reenterable	No	REENTRANT attribute tagged to a PROC allows it to be interrupted without destroying partial computation.	No
2.2.6 Pure Procedure	Implementation specific	Implementation specific	Implementation specific

Data Element Types, Groups and Operations

3.0	JOVIAL (J3)	SPL/J6	CLASP
3.1 Types of Data Elements			
3.1.1 Scalar			
3.1.1.1 Arithmetic	See 1.4.1.1.1	See 1.4.1.1.1	See 1.4.1.1.1
3.1.1.1.1 Integer	Yes	Yes	Yes
3.1.1.1.2 Fixed Point (Mixed Number)	Yes	Yes	Yes
3.1.1.1.3 Floating Point	Yes	Yes	Yes
3.1.1.1.4 Binary	No	Yes	Yes
3.1.1.1.5 Octal	Yes	Yes	Yes
3.1.1.1.6 Decimal	Yes	Yes	Yes
3.1.1.1.7 Hexadecimal	No, Implementation Specific	Yes	Yes

Data Element Types, Groups and Operations

3.0	JOVIAL (J3)	SPL/J6	CLASP
3.1.1.1.8 Multi-Precision	Optional extension	Automatic - based on number of digits specified in floating point declaration.	Automatic - based on number of digits specified in floating point declaration.
3.1.1.1.9 Other	No	No	No
3.1.1.2 Boolean	Yes, plus Status	Yes, plus Status	Yes
3.1.1.3 List or Pointer	No	No	No
3.1.1.4 Other	location	location	location
3.1.2 Non-Scalar			
3.1.2.1 Arithmetic	See 1.4.1.1.2	See 1.4.1.1.2	See 1.4.1.1.2
3.1.2.1.1 Vector	Yes	Yes	Yes
3.1.2.1.2 Matrix	Yes	Yes	Yes
3.1.2.1.3 Complex	No	No	No
3.1.2.1.4 Other	Dual Fixed Point	No	No
3.1.2.2 Bit String	No	No	No

Data Element Types, Groups and Operations

3.0	JOVIAL (J3)	SPL/J6	CLASP
3.1.2.3 Text String	Yes	Yes	Yes
3.2 Groups of Data Elements			
3.2.1 Arrays	Yes, plus tables	Yes, plus tables	Yes
3.2.2 Hierarchical Structures	No	Yes	Yes
3.2.3 Combinations Of Above	No	Yes	Limited
3.2.4 Files	Yes, see 5.1,3.1	Yes, see 5.1.3.1	No
3.3 Operations With Data Element Types and Groups			
3.3.1 Intermingling Rules for Mixed Data Types	Mixed mode arithmetic is allowed. Care must be exercised if expressions contain Hollerith or transmission code data. See Figure 2.	Mixed mode arithmetic is allowed. Care must be exercised if expressions contain textual data. See Figure 2.	Mixed mode arithmetic is allowed. Care must be exercised if expressions contain textual data for they may only be assigned to textual variables.

Data Element Types, Groups and Operations

3.0	JOVIAL (J3)	SPL/J6	CLASP
3.3.2 Conversion Rules	Conversion is automatic across the assignment operator. (Fixed to floating, floating to fixed and fixed or floating to dual). See Figure 3.	Conversion is automatic across the assignment operator.	Covernsion is automatic across the assignment operator. (Fixed to floating). The resulting type of the right term is determined by the last operatior performed in the formula evaluation.
3.3.3 Alignment Rules	Alignment is automatic for arithmetic operations. There are implementation rules concerning truncation of values and ranges of values.	Alignment is automatic for arithmetic operations. There are implementation rules concerning truncation of values and ranges of values.	Alignment is automatic for arithmetic operations. There are implementation rules concerning truncatior of values and ranges of values.
3.3.4 Precision and Computation Rules	R (Rounded) attribute. Also, host computer dependent because of the word size. MANT, CHAR functional modifiers can be used to handle parts of floating point numbers.	R (Rounded), .S (scaling) operator, and C1...C2 range function. Host computer dependent because of the word size.	RND, LIM functions, .S (scaling) operator and TEMP attribute. Host computer dependent because of the word size.

Data Element Types, Groups and Operations

3.0	JOVIAL (J3)	SPL/J6	CLASP
3.3.5 Precedence and Sequencing Rules	Left to right scan with following operator precedence rules: Highest negation exponentiation multiplication, division addition, subtraction relational boolean Lowest Parentheses may be used to modify precedence. Nested parentheses evaluated innermost to outermost.	Left to right scan with following operator precedence rules: scaling (.S) operator negation exponentiation multiplication, division addition, subtraction logical relational boolean Parentheses may be used to modify precedence. Nested parentheses evaluated innermost to outermost.	Left to right scan with following operator precedence rules: scaling (.S) operator negation exponentiation multiplication, division addition, subtraction logical relational boolean Parentheses may be used to modify precedence. Nested parentheses evaluated innermost to outermost.
3.4 Accessibility of Data			
3.4.1 Hardware Defined	Individual bits and bytes can be accessed through the functional modifiers BIT and BYTE. These keywords are followed by two numbers that represent the initial bit or byte of the data item and the number of bits of bytes. Numeric and textual operands are necessary.	Bits and bytes by the partial index mechanism. Individual registers by the HARDWARE and INDEX declaratives. These declaratives are machine dependent and implementation specific. Indirect addressing.	Individual registers by the HARDWARE and INDEX declaratives. These declaratives are machine dependent and implementation specific. Logical operators set or reset bits and shift bits right or left (arithmetic only).

Data Element Types, Groups and Operations

3.0	JOVIAL (J3)	SPL/J6	CLASP
		Logical operators set or reset bits and shift bits right or left (logical and arithmetic depending on type of formula).	
3.4.2 Language Defined	All data variables, constants, arrays, tables, strings are accessible by commands.	All data variables, constants, arrays, tables, groups are accessible by commands.	All data variables, constants, arrays, groups are accessible by commands.
3.5 Scope of Data	See 1.3.6	See 1.3.6	See 1.3.6

	Executable Statements		
4.0	JOVIAL (J3)	SPL/J6	CLASP
4.1 Assignment, Exchange and Computation Statements			
4.1.1 Data Element Types			
4.1.1.1 Scalar	Variable = formula $ Variable == formula $	Variable = formula Variable == formula	Variable = formula Variable == formula
4.1.1.1.1 Arithmetic	Integer variables Floating Point variables Fixed Point variables See figure 3	Integer variables Floating Point items Fixed Point variables	Integer variables Floating Point variables Fixed Point variables
4.1.1.1.2 Boolean	Boolean and Status variables.	Boolean and Status variables	Boolean variables
4.1.1.1.3 Comparison	No	No	No
4.1.1.1.4 List or Pointer	No	No	No
4.1.1.1.5 Other	No	No	No

	Executable Statements		
4.0	JOVIAL (J3)	SPL/J6	CLASP
4.1.1.2 Non-Scalar	Variable = formula $ Variable == formula $	Variable = formula Variable == formula	Variable = formula Variable == formula
4.1.1.2.1 Arithmetic	Dual fixed point	List of variables vector variables matrix variables	List of variables vector variables matrix variables
4.1.1.2.2 Bit String	No	No	No
4.1.1.2.3 Text String	Hollerith or transmission code variables	Textual variables	Textual variables
4.1.2 Arrays	No	variable = formula variable == formula	variable = formula variable == formula
4.1.3 Hierarchical Structures	No	No	No
4.1.4 Nested Assignment (Factoring)	No	Yes, with multiple variables and expressions	Yes

4.0	Executable Statements		
	JOVIAL (J3)	SPL/J6	CLASP
4.1.5 Conversion Rules for Results	Arithmetic assignment statements have automatic conversion between fixed, dual fixed and floating point. If the right term is more precise, excess precision is lost, either by truncation or rounding. If the right term has less significance than the variable it is padded with leading zeros or if the right term has more significance the excess is lost to overflow. If literal values are larger leading characters are last. If literal values are smaller they are right justified and padded with blanks. More specific rules for each assignment type are given in the manual.	Arithmetic assignment statements have automatic conversions between fixed and floating point. The formula is computed with maximum precision and assigned to left term. If the left term has less precision than the result, the excess precision is truncated. In literal assignment statements constants are left justified. Each assignment type has more specific rules that it obeys and these are cited in the manual.	The resulting type of the right term is determined by the last operation performed in the formula evaluation. If type is logical, the value of the formula, bit for bit, will be assigned to the left term without conversion, regardless of the type of the left term. If the result is longer than the left term the excess bits will be truncated from the high-order end of the result before assignment takes place. If the result is shorter it will be stored in the left term right-adjusted, with zeros filling the excess high order bits. Automatic conversion (for each assignment type) results according to the rules specified in the manual.

	Executable Statements		
4.0	JOVIAL (J3)	SPL/J6	CLASP
4.2 Textual Data Handling Statements			
4.2.1 Editing	No, but functional modifiers BIT, BYTE can help the process.	No, but partial indexing is useful.	No
4.2.2 Conversion	No	No	No
4.2.3 Sorting	No	No	No
4.2.4 Comparison	No	No	No
4.3 Sequence Control and Decision Making Statements			
4.3.1 Unconditional Control Transfer			

Executable Statements

4.0	JOVIAL (J3)	SPL/J6	CLASP
4.3.1.1 No Return	By the GOTO statement followed by statement or program label (no period) and terminated by the dollar sign ($). The STOP statement followed by an optional statement label (no period) and terminated by the dollar sign. This statement sometimes causes a return to the operating system. Normal sequencing is from one statement to the next statement.	By the GOTO statement followed by statement label (no period). The STOP statement followed by an optional statement label (no period). This statement sometimes causes a return to the operating system. Normal sequencing is from one statement to the next statement.	By the GOTO statement followed by statement label (no period). The STOP statement followed by an optional statement label (no period). This statement sometimes causes a return to the operating system. Normal sequencing is from one statement to the next statement.
4.3.1.2 Return	By the procedure, function call statement of form procedure name followed by an optional list of calling parameters enclosed in ( ) pair separated by commas and terminated by dollar sign. The = separator separates input calling parameters on the left side from output calling parameters on the right. The END transfers control back to next statement after the procedure call. If it is desired to transfer control earlier, the RETURN $ statement is used. See 2.2.2. The CLOSE call statement of form CLOSE followed	By the procedure, function call statement of form optional list of output calling parameters separated by commas, optionally followed by = separator and followed by a period and procedure name. This is followed by optional list of input calling parameters separated by commas. Either of the parameter lists may be omitted. See 2.2.2. The CLOSE call statement of form period followed by a name. This statement provides a closed and parameterless subroutine. The EXIT statement returns control to the statement	By the procedure, function call statement of form period and procedure name followed by an optional list of calling parameters enclosed in ( ) pair and separated by commas. The = separator separates input calling parameters on left side from output calling parameters on the right. See 2.2.2. The CLOSE call statement of form period followed by a name. This statement provides a closed and parameterless subroutine. The EXIT statement returns control to the statement following the

4.0	JOVIAL (J3)	SPL/J6	CLASP
	by a name and terminated by the dollar sign. This statement provides a closed and parameterless subroutine. The RETURN $ statement can be used in CLOSE statements.	following the procedure, function or close call and optionally allows return of values for embedded procedure calls. The RETURN statement causes transfer of control to the EXIT statement and optionally allows values in EXIT statements to be overridden.	procedure, function or close calls. A GOTO statement must be used to transfer control to the EXIT statement if it is desired to leave earlier.
4.3.2 Conditional Control Transfer			
4.3.2.1 No Return	Conditional statements take one of two forms. The first is of the form IF Boolean-expression $ statement where the statement cannot contain an IF except within a BEGIN...END bracket. Another form of conditional statement provides alternatives, using the following format: IFEITH Boolean expression $ statement label. ORIF Boolean-expression $ statement label. ORIF Boolean-expression $ statement where labels are optional elements. Any number of alternatives can be listed, and the Boolean expressions	By the IF, THEN, ELSE, ORIF, keywords. Of following form: IF B THEN S or O END IF B THEN S or O ELSE S END IF B S or O END where B = boolean-expression S = statement O = ORIF clause or statement ENDALL facilitates closing out nested conditionals. Allowed within loops and nesting can be used to any depth. Complete decision tables implemented with CONDITIONS ACTIONS pair. See	By the IF, THEN, ELSE keywords of form: IF B S IF B THEN S IF B THEN S ELSE S where B = boolean expression S = statement. ENDALL facilitates closing out nested conditionals. Allowed within loops and can be nested. Switches are referenced by the GOTO command.

Executable Statements

4.0	JOVIAL (J3)	SPL/J6	CLASP
	are tested in sequence from the top; the statement associated with the first true Boolean-expression is executed and control is then passed to the statement following the END. Switches (SWITCH) are referenced with GOTO command. Indexed and item switches are allowed. EXECUTE followed by statement label allows that one statement to be executed.	manual. Switches (SWITCH) are referenced with GOTO command. Indexed switches are allowed. A form of the GOTO command allows label assignments to allow the same sequence of statements to be called from different parts of the program.	
4.3.2.2 Return	No	No	No
4.3.3 Loop or Index Control			
4.3.3.1 Loop Designation	The loop control statement is the FOR statement, which has the following general format: For parameter list 1 $ name. FOR parameter list 2 $...$ name. FOR parammeter list n $ BEGIN statements END Labels are optional.	Concise loop control with FOR statement. Designated by loop variables. UNTIL and WHILE can be used within loop to exit on certain test variable conditions.	Concise loop control with FOR imperative, with BY, TO used for loop setup. Designated by loop variable IF imperative can be used within loop. Exiting on certain test variables.

4.0	Executable Statements		
	JOVIAL (J3)	SPL/J6	CLASP
4.3.3.2 Range of Loops	Set of statements contained within BEGIN END pair. Loop variables are only defined within the range of the FOR statement and are limited to one letter (26 is maximum).	Set of statements occurring after initial FOR and terminated with END or ENDALL. Loop variables are always defined.	Set of statements occurring after initial FOR and terminated with END or ENDALL.
4.3.3.3 Iteration Control Mechanism	The parameter list consists of the initial, final and increment/decrement values which can be constants or legal arithmetic expressions. Nested FOR statements can occur to any depth. The innermost loops are processed first and the processing moves to the outer loops. The parameters following each separate FOR statement are varied in parallel. The ALL functional modifier refers to a table or item in the table and the looping is executed for each entry in the table. TEST functional modifiers can be used within loop to exit on certain test variable conditions or cause incrementing and testing of the loop variable.	Initial, final and increment/decrement values can be constants or legal arithmetic expressions. The final parameter can be a boolean expression. Nested FOR imperatives can occur to any depth. The innermost loops are processed first and the processing moves to the outer loops. Use of ENDALL eliminates need for multiple ends. The ALL functional modifier refers to a table or item in the table and the looping is executed for each entry in the table. The compiled instructions needed for the testing and incrementing specified in the loop statement are inserted at the end of the statement. Incrementation of the loop variables by the current value of the incrementation formulas takes place in the reverse order of that in which they are defined. The incrementation	Initial, final and increment/decrement values must be constants, or integer variables, loop variables are undefined after normal loop termination. Nested FOR imperatives can occur to any depth. The innermost loops are processed first and the processing moves to the outer loops. Use of ENDALL eliminates need for multiple ends.

	Executable Statements		
4.0	JOVIAL (J3)	SPL/J6	CLASP
		is immediately followed (if specified) by a test of the controlling loop variable. If this variable has not gone beyond the current value of its limit, control is transferred to the top of the loop; otherwise execution proceeds to the instruction following the loop statement.	
4.3.4 Real Time Control			
4.3.4.1 Multi-tasking	No	If parallel processing is available the DO and JOIN statements are provided. The DO statement specifies segments of code to be handled in parallel, while the JOIN specifies the point at which the parallel paths previously defined by last DO statement come together. The range begins with the DO and ends at the JOIN.	No
4.3.4.2 Scheduling and Dispatching	No	The WAIT statement is used in conjunction with either a conditional or loop statement and indicates that until the Boolean formula is satisfied, that statement will be repeated.	No

Executable Statements

4.0	JOVIAL (J3)	SPL/J6	CLASP
4.3.4.3 Storage Allocation			
4.3.4.3.1 Static	No	No	No
4.3.4.3.2 Dynamic	No	No	No
4.3.4.4. Access Restrictions			
4.3.4.4.1 Memory	No	The statements LOCK and UNLOCK may be used to protect or make available an area of memory on those computers that have memory protect capability. LOCK or UNLOCK is followed by identifiers specifying the range of the area to be protected.	The statements LOCK and UNLOCK may be used to protect or make available an area of memory on those computers that have memory protect capability. LOCK or UNLOCK is followed by identifiers specifying the range of the area to be protected.
4.3.4.4.2 Registers	No	No	The LOCK statement also tells the compiler that it can no longer use the indicated registers in the compiled code except where the programmer references them. If compilation canno continue without a given

Executable Statements

4.0	JOVIAL (J3)	SPL/J6	CLASP
			register, the compiler will save the programmer specified value and restore it when through using the register.
4.3.4.4.3 Interrupts	No	For interrupts, LOCK and UNLOCK inhibit or activate previously declared interrupts. The identifier that follows LOCK, UNLOCK specifies the interrupt which has been declared in ON statement.	For interrupts, LOCK and UNLOCK inhibit or activate previously declared interrupts. The identifier that follows LOCK, UNLOCK specifies the interrupt which has been declared in ON statement.
4.3.4.5 Interrupt Handling	No	The ON statement. See 4.3.5.	The ON statement. See 4.3.5
4.3.5 Error Condition and Program Checking	No	The chronic statement of form ON followed by boolean-formula or interrupt name followed by a statement. These statements are executed only when their enabling condition occurs. They are not part of the normal sequence of statement execution. The condition for executing a chronic statement	The chronic statement of form ON followed by an interrupt name. After this statement any number of statements can occur and they are terminated by the EXIT statement. These statements are to be executed upon the occurrence of a specified enabling condition, usually a hard-

4.0	JOVIAL (J3)	SPL/J6	CLASP
	Executable Statements		
		is dictated by a Boolean formula which is automatically evaluated whenever its first operand acquires a new value, either by assignment or by hardware action. This operand may be either program declared data or a hardware device. A chronic statement has an implicitly built-in return to the interrupted program sequence.	ware interrupt. Chronic statements are not part of the normal sequence of program execution. When the interrupt occurs, execution of the current task is automatically suspended and control is passed to the first statement following the ON. When the EXIT is reached control is returned to the task which was interrupted.
4.4 Symbolic Data Handling Statements			
4.4.1 Algebraic Expression Manipulation	No	No	No
4.4.2 List Handling	No	No	No
4.4.3 String Handling	No	No	No

	Executable Statements		
4.0	JOVIAL (J3)	SPL/J6	CLASP
4.4.4 Pattern Handling	No	No	No
4.5 Interaction With Operating System and/or Environment Statements			
4.5.1 Input/Output			
4.5.1.1 File Initialization/ Processing/Termination	Form of OPEN INPUT F, INPUT F, SHUT INPUT F where F is name of file. A file may be read, one record at a time, by the execution of a series of input statements. The first statement executed in such a series must be an OPEN INPUT statement, which activates the file, and the last must be a SHUT INPUT statement, which deactivates it. Input records so read may be variables, entire arrays, entire tables, sequences of table entries, or individual table entries. A read operation transfers	The file initialization statements of SPL are OPEN and SHUT. OPEN initializes the file being operated upon, while SHUT signals that file will no longer be used. The initialization that OPEN performs is a function of the computer and the operating environment in which the SPL program operates. It may set system flags or initiate buffering, or any other one-time function required. Conversely, SHUT may reset system flags, close buffers, write an end-of-file on outputs, or execute any	No

4.0	JOVIAL (J3)	SPL/J6	CLASP
	the string of bits or bytes comprising a file record from the file into the computer's internal memory, to represent the value or values of a designated input record. A read is terminated when the entire input record has been represented. (Any bits or bytes left in the file record go unread and are skipped over.) A read is also terminated when the string of bits or bytes of the file record is exhausted. (The remainder of the input record, if any, is undefined.) An OPEN INPUT statement activates an inactive file and initializes its position to zero. A SHUT INPUT statement deactivates an active file. Any input statement that designates an input record initiates a read operation that will transfer a record from the file into the computer's internal memory, thus incrementing file-position by one. Form of OPEN OUTPUT F, OUTPUT F, SHUT OUTPUT F where F is name of file. A file may be written, one record at a time, by the execution	system accounting required. READ and WRITE input or output a declared file. The READ imperative brings data from some external device into a defined place in core. Conversely, WRITE passes data from some defined place in core to an external device in the form and manner specified by the file declaration. The nature of this data has been defined in the file declaration and is independent of the actual device used. Because of this, the file declaration specifies a logical rather than a specific physical device and the actual logical-physical device equation is done at execution time.	

4.0	JOVIAL (J3)	SPL/J6	CLASP
	of a series of output statements. The first statement executed in such a series must be an OPEN OUTPUT statement, which activates the file, and the last must be a SHUT OUTPUT statement, which deactivates it. Output records so written may be numeric or literal constants, variables, arrays, etc. A write operation transfers the string of bits or bytes representing a designated output record from the computer's internal memory out onto the file, as a file record. A write is terminated when the entire output record has been transferred. (For rigid record-length files, the remainder of the file record, if any, is undefined.) A write is also terminated when the number of bits or bytes transferred equals the declared maximum file-record size. An OPEN OUTPUT statement activates an inactive file and initializes its position to zero. A SHUT OUTPUT statement deactivates an active file. Any output statement that designates		

4.0	Executable Statements		
	JOVIAL (J3)	SPL/J6	CLASP
	an output record initiates a write operation that will transfer a record from the computer's internal memory out onto the file, thus incrementing file-position by one.		
4.5.1.2 File Positioning and Handling	A JOVIAL file is a self-indexing storage device, meaning that the record available for transfer to or from the file depends on the file's current position. The records of an n-record file are indexed from 0 through n-1, and the index of the record currently available for transfer is designated, as file-position, with the functional modifier POS, operating on the name of an active file. File position ranges from 0 (indicating "rewound") through n (indicating "end-of'file"). The transfer of a record to or from a file automatically increments the file position by one.	REWIND, BACKSPACE, SEARCH, EJECT, and SKIP are the file positioning statements of SPL. They control the position of the named file. REWIND positions to the beginning, BACKSPACE moves the named file back one record, and SEARCH starts the file looking for a particular record number. EJECT is used primarily to position to the top of a new page for printing, and SKIP's primary function is to space one print line. Since these file imperatives are independent of physical devices, they may be applied to devices for which they have no meaning.	No

	Executable Statements		
4.0	JOVIAL (J3)	SPL/J6	CLASP
4.5.2 Library Reference	No, available through procedure call. See Figure 4.	No, available through procedure call. See Figure 4.	No, but could be provided through procedure call. See Figure 4.
4.5.3 Debugging	No	See 6.5.	See 6.5.
4.5.4 Storage and Segmentation Allocation			
4.5.4.1 Static	No	No	No
4.5.4.2 Dynamic	No	No	No
4.5.5 Operating System and Machine Dependent	STOP imperative - however, it is implementation specific. The compool can be thought of as an interface.	STOP, ON (Chronic), LOCK, UNLOCK, DO, JOIN, WAIT imperatives. The compool can be thought of as an interface.	STOP, ON (Chronic), LOCK, UNLOCK imperatives.
4.5.6 Others	Language is oriented to binary type computers.	Language is oriented to binary type computers.	Language is oriented to binary type computers.

Non-Executable Statements

5.0	JOVIAL (J3)	SPL/J6	CLASP
5.1 Data Declarations			
5.1.1 Data Element Type Declarations			
5.1.1.1 Constants			
5.1.1.1.1 Arithmetic			
5.1.1.1.1.1 Hexadecimal	Implementation specific.	X or HEX 'HEX-STRING'	X 'HEX -DIGITS'
5.1.1.1.1.2 Decimal	Integer Floating Point Fixed Point Dual Fixed Point	Integer Floating Point Fixed Point	Integer Floating Point Fixed Point
5.1.1.1.1.3 Octal	O (OCTAL-DIGITS)	O or OCT 'OCTAL-DIGITS'	O 'OCTAL-DIGITS'
5.1.1.1.1.4 Binary	No	B or BIN 'BINARY-INTEGERS'	B 'BINARY-DIGITS'
5.1.1.1.2 Boolean	0,1, and Status constant of form V (letter or name).	TRUE, FALSE ON, OFF 0,1 V(TRUE) or V(FALSE) V(ON) V(OFF) Status constant of form V (identifier) or identifier.	TRUE, FALSE ON, OFF

Non-Executable Statements

5.0	JOVIAL (J3)	SPL/J6	CLASP
5.1.1.1.3 Textual	Decimal integer specifying number in character string plus H or T preceding character string with character string enclosed by parentheses.	'CHARACTER STRING' optionally preceded by decimal integer specifying number in character string.	'CHARACTER STRING' optionally preceded by decimal integer specifying number in character string.
5.1.1.1.4 Other	Location constant of form L (address).	Location constant of form: LOC 'numeric formula ' or statement label.	Location constant of form L 'item name or ' statement label.
5.1.1.2 Variables	The ITEM keyword is used to define data items with their associated identifiers and to describe the data types and data attributes affecting their subsequent manipulation; form is ITEM identifier-type-attributes where attributes are optional. Item declarations must be terminated with the dollar sign ($). A MODE declaration is followed by an item description which declares an implicit mode of definition for subsequent simple item names that are otherwise undefined. The $ terminates the declaration. The MODE is effective until superceded by another MODE declaration.	The ITEM keyword is used to define data items with their associated identifiers and to describe the data types and data attributes affecting their subsequent manipulation; form is ITEM identifier-type-attributes where attributes are optional. A MODE declaration is followed by an item description which declares an implicit mode of definition for subsequent simple item names that are otherwise undefined. The MODE is effective until superceded by another MODE declaration.	The DECLARE keyword is used to define data items with their associated identifiers and to describe the data types and data attributes affecting their subsequent manipulation; form is DECLARE type-attributes, identifier.

Non-Executable Statements

5.0	JOVIAL (J3)	SPL/J6	CLASP
5.1.1.2.1 Arithmetic	fixed point (A) floating point (F) integer (I) dual fixed point (D) Fixed point and integer items can be followed by following attributes: S, or U, or R, and number of bits. Fixed items allow number of bits right of the binary point. Floating point items can be followed by the R attribute. Dual fixed point items are followed by the following attributes: number of bits per half, S or U, signed number of fractional bits per half or R. E is allowed to express exponents or powers of numbers.	fixed point (A or FIXED) floating point (F or FLOATING) integer (I or INTEGER) These arithmetic items may be declared to any of four number systems - decimal (assumed), binary (BIN), octal (OCT), and hexadecimal (HEX). Fixed point and integer items can be followed by following attributes: S or SIGNED, U or UNSIGNED, R or ROUND, C1...C2 (range values) and number of bits. Fixed items allow number of fractional bits. Floating point items can be followed by the following attributes: R or ROUND, number of decimal places to the right of the decimal point. E is allowed to express exponents or powers of numbers. The word CONSTANT in the item declaration indicates item cannot be changed. LOCAL provides a means for declaring in the main program the item declarations which are normally declared in procedures. The keyword DECLARE is discussed in section 5.1.1.3.	fixed point (A or FIXED) floating point (F or FLOATING) integer (I or INTEGER) Integer declarations can be followed by the attributes CONSTANT for items whose values cannot be changed, PARAMETER whose values do not change but are not known until execution time. Fixed point items can be followed by number of bits, number of fractional bits, CONSTANT, PARAMETER, TEMP which prevents the compiler from rescaling items that are temporary values having many different scalings. Floating point declarations can be followed by attribute concerning number of decimal places, CONSTANT and PARAMETER. E is allowed to express exponents or powers of numbers.

Non-Executable Statements

5.0	JOVIAL (J3)	SPL/J6	CLASP
5.1.1.2.2 Boolean	B and/or S followed by status list where previously used identifiers must be enclosed in parentheses and preceded with a V. Number of bits may also be specified. Otherwise compiler generates minimum number.	B or BOOLEAN and/or S or STATUS followed by status list where previously used identifiers must be enclosed in parentheses and preceded with a V. Number of bits may also be specified. Otherwise compiler generates minimum number.	B or BOOLEAN followed by CONSTANT or PARAMETER.
5.1.1.2.3 Textual	H (Hollerith) or T (transmission code)	TEXT followed by optional number of bytes, character field attributes, number of fractional bytes and status list.	T or TEXT optionally followed by number of bytes, CONSTANT, PARAMETER.
5.1.1.2.4 Other	No	No	No
5.1.1.3 Presetting of Declarations	P followed by JOVIAL constant.	All simple items allow separator = to follow description to be followed with an SPL constant.	All simple items allow separator = to follow description to be followed with a CLASP constant.
5.1.1.4 Nesting (Factoring) of Declarations	No	The keyword DECLARE is used for factored item declarations because it is the item descriptions that determine the nature of the declaration. DECLARE is followed by all data attributes. Commas	Yes

Non-Executable Statements

.0	JOVIAL (J3)	SPL/J6	CLASP
		separate the different declarations. Presetting can occur within factored declarations.	
5.1.1.5 Default Options	No	No	No
5.1.1.6 Numbering Conventions	Bits, bytes of word are numbered consecutively from left to right (MSD-LSD) and begin at zero. Numbers are assumed decimal unless indicated otherwise.	Bits, bytes of word are numbered consecutively from left to right (MSD-LSD) and begin at zero. Numbers are assumed decimal unless indicated otherwise.	Bits, bytes of word are numbered consecutively from left to right (MSD-LSD) and begin at zero. Numbers are assumed decimal unless indicated otherwise.
5.1.2 Group Type Declarations			
5.1.2.1 Array Declarations	The keyword ARRAY followed by an identifier,or list of dimension numbers, an item description and terminated by the $. The first integer of the dimension list designates the number of rows, second, the number of columns, third the number of planes, etc. The item description declares the	The keyword ARRAY followed by an identifier followed by integer or integers separated by commas and enclosed in parentheses. The first integer indicates number of rows, the second, the number of columns, the third, the number of planes, the fourth, number of volumes, etc. It is then	The format for array declarations is an extension of that for data element type declarations. The form is DECLARE type-attributes, identifiers followed by a dimension list of integer constants enclosed in parentheses, separated by commas and can get no larger than three dimensions. The

5.0	JOVIAL (J3)	SPL/J6	CLASP
	array's type. There are compiler allocated tables of form TABLE followed by identifier followed by V (variable) or R (rigid) number of entries and P (parallel) or S (serial). Items declarations are enclosed in BEGIN END brackets. Programmer allocated tables have the same format except that the number of words/entry precedes the P or S attribute. In addition the programmer allocated item statement declares the word number, the bit number and whether the packing is N (normal), M (medium), or D (dense). Some implementations allow B for byte. The STRING declaration is part of the TABLE declaration and allows an item not just once per entry, but many times per entry. Tables can be duplicated with the L attribute and a suffix added to table name.	optionally followed by the data type and attributes. An optional repetition factor is included. Elements of arrays must be reached through subscripting. See 1.3.5.1. A variable dimension array is specified by replacing one or more of the integer constants in the dimension list with the names of simple integer items. The TABLE declaration declares a serial type table and the DECLARE table declaration is a parallel type table and they are created by factoring array declarations. The serial TABLE declaration contains an identifier, number of words/entry, number of entries and optional types and attribute information. All the data element attributes are valid for table declarations.	first integer constant is the number of rows, then columns and then planes. An optional repetition factor is included. A variable dimension array is specified by replacing one or more of the integer constants in the dimension list with the names of simple integer items. The size of the variable dimension is the value of the integer at procedure execution time. Implicit subscripts allowed by including in the array's dimension list the names of integer variables that are to be implicitly defined as subscripts for that array.

Non-Executable Statements

5.0	JOVIAL (J3)	SPL/J6	CLASP
5.1.2.2 Hierarchical Structures	No	A group is declared by labeling a data declaration, item, array,table, declare. Hierarchical structures can be built out of these declarations. The qualified name is a sequence of identifiers specified left to right in order of structure complexity - group, table, item. The names are separated by the simple prime with no blanks permitted between names and primes.	A limited form of groups is declared by labeling a data declaration statement and form is identifier.data declaration. The elements of the group can have different attributes.
5.1.2.3 Procedure, Function, Subroutine Declarations	Yes, See 4.3.1.2	Yes, See 4.3.1.2. Also these declarations allow attributes RECURSIVE, REENTRANT, INLINE and ENTRANCE which affect how the procedure works.	Yes, See 4.3.1.2. Also these declarations allow attribute INLINE which causes macro substitution.
5.1.2.4 Presetting	Yes, part of BEGIN END brackets.	Yes, PRESET declaration for arrays, and another method for tables.	Yes
5.1.2.5 Nesting (Factoring) of Declarations	No	Yes	Yes

.0	Non-Executable Statements		
	JOVIAL (J3)	SPL/J6	CLASP
5.1.2.6 Default Options	No	No	No
5.1.2.7 Numbering Conventions	Words, entries of tables, array elements are numbered from zero to number of /words/entries/array element/-1.	Words, entries of tables, array elements are numbered from zero to number of /words/entries/array elements/-1.	Words, entries of tables, array elements are numbered from zero to number of /words/entries/ array element/-1.
5.1.3 Interaction with Operating System and/or Environment			
5.1.3.1 File Declarations	A file is a list of records, which are themselves strings of bits or of 6-bit, Hollerith-coded or 8 bit EBCDIC coded bytes. A file's records are either all binary (B) or all Hollerith (H), and they are generally homogeneous in size, content, and format. (When heterogeneous records are organized into a file, the program must provide for distinguishing among them.) Record format is not described in the file declaration, however, since it is	After the words DECLARE FILE comes the file-name, followed by the set of attributes to be associated with that file. After all related attributes are listed, the data in the file are described in a manner similar to a table declaration. This data can be used directly in formulas or assigned to other variables of a different mode within the program, and all such conversions are implicit. Multiple file declarations may appear within one	No

5.0	JOVIAL (J3)	SPL/J6	CLASP
	determined by the input or output records in the statements that read or write the file. In the file declaration, both number of records and number of bits or bytes per record may be estimated maximums. The listed status constants are associated with the file name and denote the possible states of the storage device containing the file. The storage-device name indicates, in compiler-dependent terms, the particular storage device containing the file. Attribute list is as follows: H = Hollerith B = Binary R = Rigid length V = Variable length	DECLARE and always begin with the word FILE, followed by the file-name, the attribute-list, and a description of the data elements within the file. Attribute list is as follows: DEVICE = identifier MODE = BINARY or B, BCD or H ASCII or EBCDIC EOF = statement-label ERROR = statement-label KEY = identifier NOTFND = statement-label VARIABLE = identifier BLOCKS = decimal integer SIZE = decimal integer, hex constant	
5.1.3.2 Storage and Segmentation Declarations	An OVERLAY declaration, by allocating blocks of storage space, indicates the arrangement, in memory, of previously-declared items, arrays, and tables. The format of an OVERLAY declaration is the declarator OVERLAY followed by an	The OVERLAY declaration positions previously declared simple items, tables, or arrays within a program and, optionally, assigns a name to this group which may be used in nonscalar operations. Through this declarator, the	The OVERLAY declaration is used to indicate the order in which previously declared simple items or arrays should appear in the object program and the location in which they should be stored in memory. It is also used to assign two or

.0	JOVIAL (J3)	SPL/J6	CLASP
	overlay list. Each block listed in an overlay declaration is allocated storage beginning at a common, though undefined, origin. Each block thus "overlays" the other blocks listed in the declaration. A name may appear only once in an overlay declaration, but may appear in other overlay declarations if logical inconsistencies are avoided.	programmer can indicate the order in which his tables, arrays, and items should appear in the program. The declarator also enables the programmer to have two or more simple items, tables, or arrays assigned to the same location. All nonabsolute overlaid data will occur in the order in which the OVERLAY declarations are encountered. The nonoverlaid data is assigned after the overlaid data. The format of an OVERLAY declaration is the declarator OVERLAY followed by an overlay list.	more such items or arrays to the same memory location, enabling economical use of storage to be made or a single data word to be treated as having different data types or attributes. The format of an OVERLAY declaration is the declarator OVERLAY followed by an overlay list.
5.1.3.3 Hardware Declarations	No	These declarations are machine dependent and cannot be defined for all machines. The keyword HARDWARE is followed by a name assigned to a machine register followed by a hardware code or machine address. The INDEX declaration is followed by identifiers which declare indices and priority. The main use is for compiler code optimization. The keyword INTERRUPT is mentioned but little is discussed.	These declarations are machine dependent and cannot be defined for all machines. The keyword HARDWARE is followed by a name assigned to a machine register followed by a hardware code or machine address. The INDEX declaration is followed by identifiers which declare indices and priority. The main use is for compiler code optimization.

5.0	Non-Executable Statements JOVIAL (J3)	SPL/J6	CLASP
5.1.3.4 Format Declarations	No	No	No
5.2 Compiler Directives			
5.2.1 Optimization			
5.2.1.1 Space	No	OPTIMIZE, SPACE END where the code to be optimized is between the SPACE END brackets. See 5.1.3.3.	OPTIMIZE, SPACE UNSPACE where the code to be optimized is between the SPACE UNSPACE brackets. See 5.1.3.3.
5.2.1.2 Time	No	OPTIMIZE, TIME END where the code to be optimized is between the TIME END brackets. See 5.1.3.3.	OPTIMIZE, TIME UNTIME where the code to be optimized is between the TIME UNTIME brackets. See 5.1.3.3. SIC, UNSIC provides that no optimization is to be performed on code between SIC UNSIC brackets.
5.2.2 Debugging Aids	See 6.6.	See 6.6.	See 6.6.
5.2.3 Documentation	Implementation specific	Implementation specific	Implementation specific

5.0	Non-Executable Statements		
	JOVIAL (J3)	SPL/J6	CLASP
5.2.4 Documentation Control	Implementation specific	Implementation specific	Implementation specific
5.2.5 Storage Control	No	Size directive of form STORAGE numeric-formula which informs compiler of amount of memory available.	No
5.2.6 Compilation Control	See 6.5.1.	See 6.5.1.	See 6.5.1.
5.2.7 Others	DIRECT$ JOVIAL for inclusion of machine code between DIRECT$ JOVIAL brackets.	DIRECT END for inclusion of machine code between DIRECT END brackets. The NUMBER ORIGIN directive allows zero origin for arrays and tables to be overridden.	DIRECT END for inclusion of machine code between DIRECT END brackets.

Structure of Language and Compiler Interaction

6.0	JOVIAL (J3)	SPL/J6	CLASP
6.1 Self-Modification of Programs	No	No	No
6.2 Bootstrapping	Yes	No	No
6.3 Extensible	The bootstrapping of JOVIAL compilers allows extensions to versions of JOVIAL written in JOVIAL. However, there is no real extensibility in the language.	Moderate extensibility with DEFINE, PATTERN END, AS END declaratives which provide text substitution and macro expansion.	No
6.4 Subsets or Dialects	Basic JOVIAL	The SPL manual describes optional forms for SPL coding which allows compatibility with Basic JOVIAL. SPL itself is a dialect of JOVIAL rather than an extension.	No
6.5 Debugging, Parametric Programming Facilities			
6.5.1 Compilation Time	DEFINE declaration of form DEFINE followed by a name followed by text string enclosed in double primes and terminated by $.	The DEFINE directive of form DEFINE followed by sign-string, AS (separator) followed by signs substituted followed by END.	No

Structure of Language and Compiler Interaction

6.0	JOVIAL (J3)	SPL/J6	CLASP
6.5.2 Object Time	No, debugging procedures must be developed to provide this facility.	The TRACE,UNTRACE directives. For production type compilations, the DEFINE directive can remove debugging statements. The COUNT, END directives used to time sections of programs.	The TRACE,UNTRACE directives. The COUNT UNCOUNT directives used to time sections of the program.
6.6 Effect of Language Design On Implementation Efficiency, Remarks	Compilation time is rather long but object time efficiency is good. The language has several restrictions and is not as general as it could be. Reserved words cannot be used as identifiers. The TABLE declarations allow data to be structured for efficient storage allocation and execution. Many of the concepts are machine dependent. JOVIAL appears to be the favored language for command and control and has been a U.S. Air Force and Navy standard. Many compilers are available.	Compilation time is rather long but object time efficiency is good. The language has several restrictions and is not as general as it could be. Reserved words cannot be used as identifiers. The TABLE declarations allow data to be structured for efficient storage allocation and execution. Many of the concepts are machine dependent. The decision tables appear to be a very significant feature. The language provides options for code optimization, debugging and error checking. Recursive procedures are available although optional so that the rest of the compiler is not encumbered. Compilers are available for the UNIVAC 1824C but use the SPL Mark II version.	The first compiler was scheduled for completion during the summer of 1970. The language has many restrictions and is not as general as it could be. The language specification is vague in many areas and needs to be cleaned up. Language appears to have been designed mostly to allow compiler optimization of object code. The reserve words cannot be used as identifiers. Many of the concepts are machine dependent.

SCOPE OF NAMES FOR JOVIAL (J3)

Name or Label	Local Variable Scope (Procedure Defined)	Global Variable Scope (Main Program)	System Variable Scope
Statement Name	Prefixing to statement or formal output parameter (1)	Prefixing to a statement (1)	----
Switch Name	Switch declaration (1)	Switch declaration (1)	----
Close Name	Close declaration (1) or formal input parameter	Close declaration (1)	----
Item Name	Item (2) or mode (3) declaration	Item (2) or mode (4) declaration	Compool
Array Name	Array declaration (2)	Array declaration (2)	Compool
Table Name	Table declaration (2)	Table declaration (2)	Compool
Table-Item Name	Table declaration (2)	Table declaration (2)	Compool
File Name	File declaration (2)	File declaration (2)	----
Procedure Name	----	Procedure declaration	Library

Notes:
(1) If defined within a FOR statement, name must not be referenced from outside the innermost FOR statement containing definition.
(2) Definition must precede the first reference to the name.
(3) If not defined in main program by earlier declaration, nor in compool, then defined at first appearance by last previous MODE declaration.
(4) If not defined in main program or this procedure by earlier declaration (mode included), nor in compool, then defined by last previous MODE declaration.

Figure 1

RESULTS FROM MIXED MODE OPERATIONS

	INTEGER (I)	FIXED (A)	FLOATING (F)	BOOLEAN (B)	STATUS (V)	TEXT (T or H)
INTEGER	I	A	F	I	I	I
FIXED	A	A	F	A	A	A
FLOATING	F	F	F	F	F	F
BOOLEAN	I	A	F	I	I	I
STATUS	I	A	F	I	I	I
TEXT	I	A	F	I	I	T,H

Figure 2

JOVIAL (J3) CONVERSION RESULTS (ARITHMETIC OPERATIONS)

	Types of Item, Constants	Arithmetic Operators	Type of Result
1.	Integer and Integer	+,-,*	Integer
2.	Integer and Integer	/	Fixed Point*
3.	Integer and Fixed Point	+.-,*,/	Fixed Point
4.	Integer and Floating Point	+,-,*,/	Floating Point
5.	Fixed Point and Fixed Point	+,-,*,/	Fixed Point
6.	Fixed Point and Floating Point	+,-,*,/	Floating Point
7.	Floating Point and Floating Point	+,-,*,/	Floating Point

*A Type Item Declaration

Figure 3

Library Procedures

	Jovial (J3)	SPL/J6	CLASP
1. Conversion Procedures			
A) Input			
1) Convert EBCDIC to ASCII	✓		
2) EBCDIC decimal to floating point	✓		
3) EBCDIC hexadecimal to unsigned binary	✓		
4) EBCDIC integer to signed binary	✓		
5) EBCDIC octal to unsigned binary	✓		
B) Output			
1) Convert ASCII to EBCDIC	✓		
2) Convert signed fixed-point number to EBCDIC	✓		
3) Convert floating-point number to EBCDIC	✓		
4) Convert integer to EBCDIC hexadecimal	✓		
5) Convert integer to EBCDIC decimal	✓		
2. Mathematical Procedures			
A) Trigonometric			
1) Sine	✓		✓
2) Cosine	✓		✓
3) Tangent	✓		✓
4) Cotangent	✓		✓
5) Arc-sine	✓		✓
6) Arc-cosine	✓		✓
7) Arc-tangent	✓		✓

Figure 4

	JOVIAL(J3)	SPL/J6	CLASP
B) General			
1) Hyperbolic sine	✓	✓	
2) Hyperbolic cosine	✓	✓	
3) Hyperbolic tangent	✓	✓	
4) Natural logarithm	✓	✓	
5) Common logarithm	✓	✓	
6) Compute e^x	✓		
7) Square root (positive)	✓	✓	
11) Integer divide	✓		
3. General Procedures			
A) Sort Single Word Entries	✓		
B) Sort Multiple Word	✓		
C) Compare Two Fields			

Group III LANGUAGES

INTRODUCTION

In 1958 a language known as the International Algebraic Language (IAL) appeared. It was the product of both American and European computer users, computer manufacturers and universities. The language was intended to be a standard and as such described computational processes and did not provide input/output facilities. IAL subsequently became known as ALGOL and then ALGOL 58. However, there were weaknesses in this language and it was improved considerably, and published as ALGOL 60. A revised ALGOL 60 report was issued in 1962 which corrected known errors, and eliminated ambiguities in ALGOL 60. ALGOL is primarily a scientific/numeric programming language and does not allow processing of character or bit strings. It has not been popular in the United States because the character set did not exist on any computer input devices. Many subsets and extensions resulted from the different ALGOL's (JOVIAL, NELIAC).

FORTRAN (Formula translation) was developed in 1957 for the IBM 704 computer system. In 1958 an improved version of the language known as FORTRAN II was released. The current version, FORTRAN IV was introduced in 1962 and it was much improved over FORTRAN II. Subsequently, the languages were standardized and became known as Basic FORTRAN and FORTRAN. FORTRAN has become very popular in the United States and is primarily a scientific/numeric programming language which does not allow processing of character or bit strings. Some computer manufacturers have developed extensions to FORTRAN which allow character and bit string processing (FORTRAN V). FORTRAN is somewhat encumbered by the fact that it maintains many features of the early developed Basic FORTRAN.

The MAC language was developed at the MIT Instrumentation Laboratory for use in dynamics and automatic control theory. The first compiler was developed in 1957-1958 for the IBM 650 computer system. A revised version of the IBM 650 compiler generated code for the IBM 704 and was coupled with a 704 run time package. Later, a 650/7090 package was developed. The language was improved and implemented on the Honeywell 800, 1800 computer systems. A compiler was also implemented for the CDC 3600 for NASA MSC. MAC-360 is implemented on the IBM System/360 Model 75 computer. It is primarily a scientific/numeric language and does not allow processing of character or bit strings. It features a three-line input format which corresponds to ordinary algebraic notation. Because of this feature, the language promotes readability while allowing a powerful notation for complex algebraic expressions. The MAC language has seen limited use in the United States.

	Form of Language		
1.0	ALGOL 60	FORTRAN IV	MAC 360
1.1 Character Set			
1.1.1 Upper Case Letters	26 upper case letters of English alphabet - A to Z	26 letters of English alphabet - A to Z	26 letters of English alphabet - A to Z
1.1.2 Lower Case Letters	26 lower case letters of English alphabet - A to Z	No	No
1.1.3 Arabic Numerals	10 Arabic numerals - 0 to 9	10 Arabic numerals - 0 to 9	10 Arabic numerals - 0 to 9

Form of Language

1.0	ALGOL 60	FORTRAN IV	MAC 360
1.1.4 Special Marks or Signs	Delimiters as follows: Arithmetic operators (6) (+ - x / ÷ ↑) Relational operators (6) (< ≤ = > ≥ ≠) Boolean operators (5) (≡ ⊃ ¬ ∧ ∨) Sequential operators (6) (GOTO, IF, THEN, ELSE, FOR, DO) Separators (11) (, . 10 : ; := STEP, UNTIL, WHILE, COMMENT) Brackets (8) ([ ] ' ' BEGIN END) Declarators (7) (OWN, BOOLEAN, REAL, ARRAY, INTEGER, SWITCH, PROCEDURE) Specificators (3) (STRING, LABEL, VALUE) and Logical constants (TRUE, FALSE)	12 special marks as follows: + (plus sign) - (minus sign) * (asterisk) / (slash) $ (dollar sign) b (blank, space) . (period, decimal point) , (comma) = (equal sign) ((left parenthesis)) (right parenthesis) ' (apostrophe or single prime) BCD and EBCDIC are acceptable	b (blank or space) + (plus sign) - (minus sign) * (asterisk) / (slash) . (period or decimal point) , (comma) = (equal sign) ((left parenthesis)) (right parenthesis) ¢ (cent) F > (greater than) & (ampersand) : (colon) F $ (dollar sign) F ; (semicolon) F % (percent) F ? (question mark) F < (less than) # (pound sign) F @ (at sign) F ' (single prime) F " (double prime) F (NOTE: signs with F are legal only in Format statement.)

Form of Language

1.0	ALGOL 60	FORTRAN IV	MAC 360
			EBCDIC and Honeywell set of DCD are acceptable.
1.2 Types of Basic Elements			
1.2.1 Language Defined			
1.2.1.1 Keywords or Primitives	No, all symbolics which would intuitively be considered keywords are defined as single characters.	ASSIGN .AND. BACKSPACE BLOCK DATA CALL COMMON COMPLEX CONTINUE DATA DEFINE FILE DIMENSION DO DOUBLE PRECISION END END FILE ENTRY .EQ. EQUIVALENCE ERR EXTERNAL .FALSE. FIND FORMAT FUNCTION .GE. GO TO .GT. IF IMPLICIT INTEGER .LE. LOGICAL .LT. NAMELIST .NE. .NOT. .OR. PAUSE PRINT PUNCH READ READ INPUT TAPE READ TAPE	ABS ABVA ADJ AND ARCCOS ARCCOSH ARCSIN ARCSINH ARCTAN ARCTANH BLANK CALL CALL (NONMAC) CALL (NONMAC, RESIDENT) CALLFILE COMMBLK COMMON COS COSH COT COTH CSC CSCH DEGTORAD DET DIFEQ DIGITB DIGITG DIMENSION DO DO TO DQPHASE E EXIT EXP FILE READ FILE WRITE FLUSH FORMAT GO TO GOREQ GRTHN HDG

Form of Language

1.0	ALGOL 60	FORTRAN IV	MAC 360
		REAL RETURN REWIND STOP SUBROUTINE TO .TRUE. WRITE WRITE OUTPUT TAPE WRITE TAPE MATHEMATICAL FUNCTION SUBPROGRAM ENTRY NAMES (86)	IDMATRIX IF INDEX INDICES INTEGER INVERSE LOG LONG FORMAT LOREQ LSTHN MCDET MINIMUM MSG NEG NEW BLOCK NORZ NOTEQ NZ OR OTHERWISE PI PNZ POS PRINT PRINT FORMAT PRINT HDG PRINT MSG PROD PUNCH PUNCH FORMAT PUNCH HDG PUNCH MSG
			RADTODEG RANDOM RDFLD RDPTB RDPTG READ RELEASE RESERVE RESUME RETURN RNDMN RNDMR ROUND SATISFY SEC SECH SET SET FILE READ SET FILE WRITE SGN SIGF SIGN SIN SINH SKIP SPO SP1 SP2 SP3 SP4

Form of Language

1.0	ALGOL 60	FORTRAN IV	MAC 360
			SQRT START AT SUBROUTINE SUM TAN TANH TERMINATE TIMEOFDAY TO TR TRANSMIT TRANSPOSE TRUNCATE TYPE FORMAT TYPE HDG TYPE MSG TYPE UNDO UNIT ZERO (FILE READ) (FILE WRITE) + - = / * . , ( ) < <= > >=

Form of Language

1.0	ALGOL 60	FORTRAN IV	MAC 360
1.2.1.2 Operators			
1.2.1.2.1 Physical Graphics	+ - x / ÷ ↑ < ≤ = ≥ > ≠ ≡	+ - * / **	+ - * . / b < <= = > >=
	IF THEN ELSE FOR GO TO DO		
1.2.1.2.2 Keywords	No	.LT. .LE. .EQ. .GT. .GE. .NE. .AND. .OR. .NOT.	LSTHN LOREQ NOTEQ GOREQ GRTHN ZERO NZ PNZ NORZ POS NEG AND OR

1.0	Form of Language: ALGOL 60	FORTRAN IV	MAC 360
1.2.1.3 Punctuation			
1.2.1.3.1 Brackets			
1.2.1.3.1.1 Physical Graphics	( ) [ ] ' ' BEGIN END	( ) ; ; / /	( )
1.2.1.3.1.2 Keywords	No	(No "start") END	(No "start") START AT
1.2.1.3.2 Separators			
1.2.1.3.2.1 Physical Graphics	:= ; : , 10 . STEP UNTIL WHILE COMMENT	= , b .	= , b .

1.0	Form of Language: ALGOL 60	FORTRAN IV	MAC 360
1.2.1.3.2.2 Keywords	No	TO	TO
1.2.2 Programmer Defined			
1.2.2.1 Identifiers	Yes	Yes	Yes
1.2.2.2 Constants (literals)			
1.2.2.2.1 Numeric	Decimal integer and mixed number	Integer-Decimal Floating Point Decimal (Single, Double Precision) Complex (Floating Point Decimal)	Integer-Decimal Floating Point (Double Precision)
1.2.2.2.2 Textual	Yes	Yes	Yes
1.2.2.2.3 Boolean	TRUE, FALSE	.TRUE. and .FALSE.	No

1.0	Form of Language: ALGOL 60	FORTRAN IV	MAC 360
1.2.2.2.4 Other	No	No	Reserved constants E, PI, DEGTORAD, RADTODEG, and IDMATRIX is a dynamic constant which assumes the size of the given matrices and is one size per program.
1.2.2.3 Comments	Comments can be embedded in any statement.	Comments are constrained to comment cards only	Comments are constrained to comment cards only.
1.3 Identifier Definition			
1.3.1 Types of Identifiers			
1.3.1.1 Statement Names	Yes	Yes	Yes
1.3.1.2 Data Names	Yes	Yes	Yes
1.3.1.3 Index Variable Names	No	No	Yes

1.0	Form of Language: ALGOL 60	FORTRAN IV	MAC 360
1.3.1.4 Label Variable Names	No	Yes	Yes
1.3.1.5 Others	Switch and Procedure names	Subroutine names	Subroutine names, Derivatives
1.3.2 Formation Rules for Identifiers	Identifiers (labels and names) must consist of a letter followed by an arbitrary sequence or string of letters or numerals.	Columns 1 through 5 of the first card of a statement may contain a statement label (numeric) consisting of 1 to 5 decimal numerals (unsigned) Leading zeros are ignored. They must be unique. Symbolic names consist of a string of letters and numerals, the first of which is a letter. The string may be of any length but only the first 6 alpha-numeric characters are used. They must be unique.	Statement labels may be made up of any combination of letters and numerals plus two special marks (period and ampersand) except that this may not be simply . (period) and they may not contain embedded blanks. Identifiers are the same except period and ampersand cannot be used. The maximum number of characters is 63. They must be unique. Index variables must be one letter.

Form of Language

1.0	ALGOL 60	FORTRAN IV	MAC 360
1.3.3 Use of Reserved Words	It is recommended that the following identifiers be reserved for standard mathematical functions: ABS ARCTAN COS ENTIER EXP LN SIGN SIN SQRT	Symbolic names can be the same as keywords except where their use results in an ambiguous syntax. Intrinsic functions or external functions are reserved.	Symbolic names cannot be the same as the defined keywords (includes functions). The four reserved constants (PI, E, DEGTORAD, RADTODEG) can be overridden. IDMATRIX cannot be overridden.
1.3.4 Synonyms			
1.3.4.1 Preset	No	The EQUIVALENCE statement allows two or more names to occupy the same memory area.	No
1.3.4.2 Dynamic	No	No	No

1.0	Form of Language: ALGOL 60	FORTRAN IV	MAC 360
1.3.5 Structure of Data Names			
1.3.5.1 Subscription	References to declared arrays are followed by subscript expressions separated by commas and enclosed in [ ] pair. The number of indices in a subscripted variable must be equal to the dimension of the array. The value of the index must belong to the interval defined by means of the lower and upper bounds of the array declaration.	References to arrays are followed by simple subscript expressions separated by commas and enclosed in ( ) pair. The number of simple subscript expressions must equal the dimensions of the array. Array subscripts are ordered by rows, columns and planes.	References to arrays are followed by subscript expressions which follow rules of arithmetic expressions and are written on S lines to the right of array name. No spacing is needed between operands on the S line. Commas are used to separate extensions from index expression. Language allows a subscripted DIFEQ which is discussed later.
1.3.5.1.1 Dimensions	1 to N dimensions	1 to 3 dimensions	1 dimension
1.3.5.1.2 Classes	Constants, variables and general arithmetic expressions.	Integer constants and/or variables as simple expressions of form: integer times an integer variable ± an integer constant.	Index variables, simple literals, operations of +, -, 1 using rules of general arithmetic expressions.

1.0	Form of Language: ALGOL 60	FORTRAN IV	MAC 360
1.3.5.1.3 Form	Entier formation approximates to nearest integer.	No	Rounded to nearest integer. It must be non-negative and less than size of array.
1.3.5.1.4 Data Allowable Data Elements	Real, integer, and Boolean variables.	Integer and floating point variables.	Floating point variables (double precision).
1.3.5.2 Qualification	No	No	No
1.3.5.3 Qualification and Subscription	No	No	No
1.3.5.4 Numbering Conventions	All elements of aggregates are numbered consecutively from zero to number of elements minus one.	All elements of aggregates are numbered consecutively from one to number of elements.	All elements of aggregates are numbered consecutively from zero to number of elements minus one.

1.0	Form of Language: ALGOL 60	FORTRAN IV	MAC 360
1.3.6 Scope of Names Based on or Relative to Program Structure			
1.3.6.1 System Names	No	Labeled and unlabeled COMMON, combination of COMMON and EQUIVALENCE.	No
1.3.6.2 Global Names (Shared)	Variables in the outer block have meaning in the inner block but not vice versa.	Part of COMMON declarative; scope of data names in entire program but must be declared as COMMON in all routines that access it. Dummy argument names in subroutine definitions can be duplicated.	Scope of data names does not extend beyond the program. One must not exceed 65K words of memory. Other MAC programs can refer to data (not by name) contained in the COMMON block.
1.3.6.3 Local Names	Enclosed by BEGIN END delimiters and define a block. OWN declarations preserve values needed for later entrances.	Part of COMMON declarative; subroutine and function names are global but all variables and arrays declared in a subroutine, function or main program are local to that block.	Yes

1.0	Form of Language ALGOL 60	FORTRAN IV	MAC 360
1.4 Definition and Usage of Other Basic Elements			
1.4.1 Operators			
1.4.1.1 Arithmetic Operators			
1.4.1.1.1 Scalar	+ (addition) - (subtraction) x (multiplication) / (division) (integer division) (exponentiation)	+ (addition) - (subtraction) / (division) * (multiplication) ** (exponentiation)	+ (addition) - (subtraction) b (multiplication) / (division) E line provides exponentiation with arithmetic expressions that are confined to the E line and to the right of the operand.
1.4.1.1.2 Non-Scalar	No	No	b Multiplication (scalar-vector or vector-scalar, scalar-matrix or matrix-scalar, vector-matrix or matrix-vector, vector outer, matrix-matrix)

1.0	Form of Language ALGOL 60	FORTRAN IV	MAC 360
			* (vector cross product) . (vector dot product) / division (vector-scalar, matrix-scalar) + (vector, matrix addition) - (vector, matrix subtraction) E line containing signed numeral (-1) provides matrix inversion.
1.4.1.2 Comparison Operators			
1.4.1.2.1 Binary	< (less than) ≤ (less than or equal) = (equal) > (greater than) ≥ (greater than or equal) ≠ (not equal)	.LT. (less than) .LE. (less than or equal) .EQ. (equal) .GT. (greater than) .GE. (greater than or equal) .NE. (not equal)	< or LSTHN (less than) <= or LOREQ (less than or equal) = (equal) > or GRTHN (greater than) >= or GOREQ (greater than or equal) NOTEQ (not equal)

1.0	Form of Language		
	ALGOL 60	FORTRAN IV	MAC 360
1.4.1.2.2 Unary	By use of constant zero (0) for right part of expression.	By use of constant zero (0) for right part of expression.	ZERO (zero) NZ (non zero) PNZ (positive and non zero) NORZ (negative or zero) POS (positive or zero) NEG (negative)
1.4.1.3 Boolean Operators			
1.4.1.3.1 Binary	≡ (logical equivalence) (logical implication) (logical disjunction) (logical conjunction)	.AND. (logical conjunction) .OR. (logical disjunction)	AND (logical conjunction) OR (logical disjunction)
1.4.1.3.2 Unary	(logical negation)	.NOT. (logical negation)	No
1.4.1.4 Bit String Operators			
1.4.1.4.1 Binary	No	No	No
1.4.1.4.2 Unary	No	No	No

Form of Language

1.0	ALGOL 60	FORTRAN IV	MAC 360
1.4.1.5 Functional Modifiers	No	No	No
1.4.1.6 Others	No	No	No
1.4.2 Delimiters	Operators are same as delimiters.	No delimiters	Operators are same as delimiters.
1.4.2.1 Brackets	( ) (expression grouping, parameters) [ ] (subscripts) ' ' (literals) BEGIN END (blocking)	( ) (expression grouping, parameters, subscripts) . . (operator brackets) ' ' (textual data) / / (global, local COMMON brackets) (No "start) END (program)	( ) (expression grouping, parameters, declarations) DO TO α (statements) (No "start") START AT (program)
1.4.2.2 Separators	:= (assignment) , (parameters) (space or blank) . (indicates decimal point) ; (statement, declaration terminator) : (statement label terminator) 10 (scaling) STEP (loop parameters) UNTIL (loop parameters) WHILE (loop parameters) COMMENT (comments)	= (assignment) , (separates elements of list, subscripts) b (adjacency) . (indicates decimal point in floating point consta constants)	= (assignment) , (elements of list, statement clauses, extensions in subscripts) b (adjacency or multiplication) . (indicates decimal point in floating point constants or dot product)

1.0	Form of Language: ALGOL 60	FORTRAN IV	MAC 360
1.4.3 Punctuation	:= primarily used to designate assignment statements but also to apply an assignment concept in a FOR list and a SWITCH list. , used to separate a list of items (subscripts, parameters) ␣ . used to indicate decimal point in constants. ; separates different imperative statements or declaratives from each other. : used to identify statements by preceding them with an identifier and a colon. Used to separate identifiers from their semantic descriptions in the parameter list of a PROCEDURE declaration. Also separates upper and lower bounds of subscript values in array declarations.	= used to designate assignment. , separates lists of parameters and subscripts. b adjacency . used to indicate decimal point in constants. ( ) precede and follow arithmetic operators, passed parameters, expressions, subexpressions, and determine priority of computation.	= used to designate assignment. , separates lists of parameters, expressions, subscripts, statement clauses. b adjacency . used to indicate decimal point in constants. ( ) precede and follow expressions, file indices, parameters, and determine priority of calculation. DO TO α encloses grounds of statements.

1.0	Form of Language		
	ALGOL 60	FORTRAN IV	MAC 360
	10 decimal scaling STEP delimits loop parameters. UNTIL delimits loop parameters. WHILE delimits loop parameters. COMMENT delimits comments. ( ) precede and follow arithmetic operators, procedure parameters, expressions, subexpressions, and determines priority of computation. [ ] encloses subscript expressions. ' ' encloses literals. BEGIN END encloses block structures.		
1.4.4 Significance of Blanks	No	Blanks are somewhat significant and must not be embedded in statement labels and symbolic names. Leading blanks are ignored.	Blanks are significant and must not be embedded in identifiers. A blank sometimes signifies implied multiplication.

1.0	Form of Language ALGOL 60	FORTRAN IV	MAC 360
1.4.5 Literals	Numeric and textual which are defined as strings which can be any sequence of letters, numerals or special marks except the delimiters (' ').	Numeric and textual which are defined as strings which can be any sequence of letters, numerals or special marks except certain delimiters (' ').	Numeric and textual which are defined as character strings made up of numerals 0-9 and at most one . (decimal point). Must be contained on one line.
1.4.6 Noise Words	No	No	No
1.5 Input Format			
1.5.1 Physical Input Format			
1.5.1.1 Linear			
1.5.1.1.1 Fixed (columnar restriction)	No	Continuation specified by non-blank, non-zero, character in column 6 and no "C" in column 1. Columns 1-5 also restricted for statement labels. Up to 19 lines may be continued. Line size is 72 characters.	No

1.0	Form of Language: ALGOL 60	FORTRAN IV	MAC 360
1.5.1.1.2 String	Continuous string. There can be more than one statement per line.	No	No
1.5.1.1.3 Combination	No	No	No
1.5.1.2 Non-Linear Linear			
1.5.1.2.1 Fixed (columnar restriction)	No	No, notation does not clearly resemble statement of problem or method of solution.	Language uses a three card-per-line format (E,M,S) corresponding to the three levels-per-line of algebraic notation. Columns 1-17 restricted. Line size is 63 characters. Statements can be continued to other cards.
1.5.1.2.2 String	No, notation does not clearly resemble statement of problem or method of solution.	No	No
1.5.1.2.3 Combination	No	No	No

	Form of Language		
1.0	ALGOL 60	FORTRAN IV	MAC 360
1.5.2 Conceptual Form			
1.5.2.1 Symbolic or Formal	Highly symbolic, highly formalized, and oriented to engineering/scientific notation. Reference language is not hardware language.	Symbolic but not highly formalized and oriented to engineering/scientific notation. Reference language is hardware language.	Partially, because language is oriented to engineering/ scientific problems.
1.5.2.2 English Like	No	No	No
1.5.2.3 Pseudo English Like	No, although annotation of source program is allowed.	No, although annotation of source program is allowed.	Source program provides statements whose notation very closely resembles the problem definition (mathematical) and method of solution. Reference language is hardware language.

Structure of Program

2.0	ALGOL 60	FORTRAN IV	MAC 360
2.1 Statement Types			
2.1.1 Non-Executable Statements			
2.1.1.1 Declaratives	ARRAY BOOLEAN INTEGER LABEL OWN PROCEDURE REAL STRING SWITCH VALUE (definition)	ASSIGN TO BLOCK DATA COMMON COMPLEX DATA DIMENSION DOUBLE PRECISION EQUIVALENCE EXTERNAL FORMAT FUNCTION IMPLICIT INTEGER LOGICAL NAMELIST REAL SUBROUTINE (definition) Implicit declarations are allowed where integer variables have names beginning with I, J, K, L, M, N and any other letter implies floating point.	COMMON DIMENSION INDEX INDICES RESERVE SET TO SUBROUTINE (definition)

Structure of Program

2.0	ALGOL 60	FORTRAN IV	MAC 360
2.1.1.2 Compiler Directives	No	No, FREQUENCY was dropped from language.	NEW BLOCK REMAC PRODMAC
2.1.1.3 Comments	Immediately after the terminators ; or BEGIN it is allowed to write basic symbol COMMENT followed by arbitrary string of basic symbols which itself must be terminated by semicolon (;). For example: ; COMMENT < any sequence not containing ;> BEGIN COMMENT < any sequence not containing ;> END < any sequence not containing END or ; or ELSE> Therefore comments can go in the procedure declaration.	Comment card contains "C" in column one. Comment is then arbitrary string of letters, numerals and special marks ending in column 80.	If column 1 contains an "R" column 8 contains "9", the card is processed as a comment or if columns 9-48 are printed in print positions 81-120 of the line from previous card. If column 1 contains a "P" the card is treated as text to be printed as the first line on a new page.
2.1.2 Executable Statements			
2.1.2.1 Smallest Executable Statement	Single statement of form ABLE := 1;	Single statement of form ABLE = 1 where ABLE begins in card column 7 or beyond	Single statement of form ABLE = 1 where ABLE begins in card column 18 or beyond and an "M" is in column 1.

Structure of Program

2.0	ALGOL 60	FORTRAN IV	MAC 360
2.1.2.2 Grouped Executable Statement			
2.1.2.2.1 Block Structure	Combined individual statements form compound statements by enclosing with IN BEGIN END brackets. A program is a single block which may have nested blocks inside it.	No, to group a set of statements, one must use labels and GO TO's.	No, use of the DO TO α statement type.
2.1.2.2.2 Others	No	No	No
2.1.2.3 Loops	Loops are handled by the FOR imperative and the IF imperative.	Loops are handled by the DO imperative and by the IF imperative.	Loops are handled by the DO FOR, DO TO FOR and IF imperatives.
2.1.2.4 Procedures, Coprocedures, Functions, or Subroutines	Procedures and functions are available through the PROCEDURE END declaratives. Procedures can be nested to any depth and they permit recursion.	Statement function, intrinsic function, external function, and external subroutines.	Subroutines and external subroutines. Subroutine calls can be nested to depth of 20.

Structure of Program

2.0	ALGOL 60	FORTRAN IV	MAC 360
2.1.2.5 Inclusion of Other Languages			
2.1.2.5.1 Assembly Language (AL)	No	No	No
2.1.2.5.2 Assembly Routines	No	Machine coded. External sub-routines by use of EXTERNAL statement.	CALL (NONMAC) or CALL (NONMAC, RESIDENT) statements.
2.1.2.5.3 Higher Order Languages (HOL)	No	No	Other compilations (PL/1) can be loaded via CALL (NONMAC) or CALL (NONMAC, RESIDENT) statements.
2.1.2.5.4 HOL-AL Communications	No	No	RELEASE statement.

Structure of Program

2.0	ALGOL 60	FORTRAN IV	MAC 360
2.1.3 Intermingling Order For Non-Executable and Executable Statements	A program is a block or a compound statement which is not contained within another statement and which makes no use of other statements not contained within it. Declarations appear as a block immediately after the BEGIN symbol and can then be followed by any number of statements until the END symbol designates termination of the block. Declarations usually occur in the block head.	Declarations must precede statement function definitions which must precede the executable statements. A complete program consists of initial line, any number of statements and an END line.	Declarations (clericals) should precede executable statements. DIMENSION's should be placed before RESERVE or COMMON statements. A complete program consists of initial declarations, any number of statements and START AT α line where α is an identifier of some executable statement.
2.1.4 Operating System Interface	No	Through READ WRITE and FORMAT statements.	With READ statement, PRINT FORMAT, PUNCH FORMAT statements, FILE READ, FILE WRITE statements and RDFLD, RDPTB, ROPTG statements. MAC allows srite protection on all arrays.

Structure of Program

2.0	ALGOL 60	FORTRAN IV	MAC 360
2.2 Statement Characteristics			
2.2.1 Methods of Delimiting			
2.2.1.1 Explicit	See 1.4.3. Main delimiter is semicolon (;).	See 1.4.3.	See 1.4.3.
2.2.1.2 Implicit	No	End of unit determined by recognizing beginning of next unit. There is no initial "Start" to delimit statements forming program.	End of unit is recognized by compiler without explicit indication. There is no initial "Start" to delimit statements forming program.
2.2.2 Parameter Passage Required (Different Data Types)			
2.2.2.1 Call by Name			
2.2.2.1.1 Formal		No	No

Structure of Program

2.0	ALGOL 60	FORTRAN IV	MAC 360
2.2.2.1.1.1 Input	Array, procedure, switch names, legal expressions, variables, labels and strings.		
2.2.2.1.1.2 Output	Array, procedure, switch names, legal expressions, variables, labels and strings.		
2.2.2.1.2 Calling		No	No
2.2.2.1.2.1 Input	Array, procedure, switch names, legal expressions, variables, labels and strings.		
2.2.2.1.2.2 Output	Array, procedure, switch names, legal expressions, variables, labels and strings.		

Structure of Program

2.0	ALGOL 60	FORTRAN IV	MAC 360
2.2.2.2 Call by Value	Specification (VALUE) in procedure allows value parameter passage for those declared parameters. The formal and calling parameters are the same as call by name. Call by name is always assigned.	No	Akin to call by value but not precisely. When calling a sub-routine parameters are placed in MAC call file and no address or parameters are passed. When using CALL (NONMAC) mode values are placed in MAC call file and parameters are passed. The TRANSMIT and RESUME statements are required to put and get the data.
2.2.2.2.1 Formal			
2.2.2.2.1.1 Input			
2.2.2.2.1.2 Output			
2.2.2.2.2 Calling			
2.2.2.2.2.1 Input			
2.2.2.2.2.2 Output			

2.0	Structure of Program		
	ALGOL 60	FORTRAN IV	MAC 360
2.2.2.3 Call by Address	No	Variables, array name or array element, legal expression, subroutine name and Hollerith constant.	No
2.2.2.3.1 Formal			
2.2.2.3.1.1 Input			
2.2.2.3.1.2 Output			
2.2.2.3.2 Calling			
2.2.2.3.2.1 Input			
2.2.2.3.2.2 Output			
2.2.3 Embedding	Considerable embedding allowed with the IF statement. Boolean expressions and functions can be embedded in arithmetic expressions.	Function names can be embedded in assignment and IF statements. Boolean expressions can be embedded in arithmetic expressions.	Function names can be embedded in expressions.

Structure of Program

2.0	ALGOL 60	FORTRAN IV	MAC 360
2.2.4 Recursion	Both the use of recursive procedures and the recursive use of procedures are allowed. Procedures are always recursive and need not be declared.	No	No, although programs can call themselves but no new storage is allocated.
2.2.5 Reentrant			
2.2.5.1 Serially Reusable	Implementation specific.	Implementation specific.	Implementation specific.
2.2.5.2 Reenterable	No	No	No
2.2.6 Pure Procedure	Implementation specific.	Implementation specific.	Yes, but it is implementation specific.

Data Element Types, Groups and Operations

3.0	ALGOL 60	FORTRAN IV	MAC 360
3.1 Types of Data Elements			
3.1.1 Scalar			
3.1.1.1 Arithmetic	See 1.4.1.1.1	See 1.4.1.1.1	See 1.4.1.1.1
3.1.1.1.1 Integer	INTEGER declaration	INTEGER attribute and name beginning with I to N.	No
3.1.1.1.2 Fixed Point (Mixed Number)	No	No	No
3.1.1.1.3 Floating Point	REAL declaration	REAL attribute	No
3.1.1.1.4 Binary	No	No	No
3.1.1.1.5 Octal	No	No	No
3.1.1.1.6 Decimal	No	No	No
3.1.1.1.7 Hexadecimal	No	No	No

Data Element Types, Groups and Operations

3.0	ALGOL 60	FORTRAN IV	MAC 360
3.1.1.1.8 Multiprecision	No	DOUBLE PRECISION attribute	STANDARD
3.1.1.1.9 Other	No	No	No
3.1.1.2 Boolean	BOOLEAN declaration	LOGICAL attribute	No
3.1.1.3 List or Pointer	No	No	No
3.1.1.4 Other	LABEL specification	ASSIGN TO or label variables	SET TO or label variables
3.1.2 Non-Scalar			
3.1.2.1 Arithmetic	See 1.4.1.1.2	See 1.4.1.1.2	See 1.4.1.1.2
3.1.2.1.1 Vector	No	No	Yes
3.1.2.1.2 Matrix	No	No	Yes
3.1.2.1.3 Complex	No	COMPLEX attribute	No
3.1.2.1.4 Other	No	No	No

Data Element Types, Groups and Operations

3.0	ALGOL 60	FORTRAN IV	MAC 360
3.1.2.2 Bit String	STRING specification	No	No
3.1.2.3 Text String	STRING specification	No, Hollerith values used in assignment statements	No
3.2 Groups of Data Elements			
3.2.1 Arrays	ARRAY declaration	Yes	Yes
3.2.2 Hierarchical Structures	No	No	No
3.2.3 Combinations of Above	No	No	No
3.2.4 Files	No	Available, but there is no separate file declaration.	Available, but there is no separate file declaration.

3.0	ALGOL 60	FORTRAN IV	MAC 360
3.3 Operations With Data Element Types and Groups			
3.3.1 Intermingling Rules for Mixed Data Types	"REAL" and"INTEGER" variables can be intermingled to form arithmetic expressions. Boolean expressions can be intermingled with arithmetic expressions. Numerical and boolean expressions are handled as in FORTRAN IV. In addition, ALGOL allows relational assignments which have the value true or false and can occur wherever boolean variables can occur.	Mixed mode expressions are not allowed. Numeric and logical expressions are separate entities. The value of a numerical expressions may be of integer, real, do double precision or complex type. The type of the expression is determined by the types of its elements which are ranked as follows (lowest to highest): Integer. Real. Double Precision. Complex. The type of an expression is the type of the highest ranking element in the expression. An expression is evaluated by converting all elements to the expression type and performing all arithmetic according to this type. An integer expression is evaluated using binary integer arithmetic throughout, giving an integer value as the result. In integer arith-	There are no restrictions.

3.0	ALGOL 60	FORTRAN IV	MAC 360
		metic fractional parts arising in division are truncated, not rounded. All other calculations use binary floating point arithmetic.	
3.3.2 Conversion Rules	See FORTRAN IV column.	Conversions to higher rank are performed as follows: An integer quantity becomes the integer part of a real quantity. The fractional part is zero. A real quantity becomes the most significant part of a double precision real quantity. The least significant part is zefo. A real quantity becomes the real part of a complex quantity. The imaginary part is zero. A double precision quantity is converted to single precision and becomes the real part of a complex quantity. The imaginary part is zero. In exponentiation the types of the base and exponent are restricted such that complex exponents are not allowed, and only integer exponents may be used with complex bases. A logical expression is a sequence of logical elements separated	There are no conversions.

3.0	ALGOL 60	FORTRAN IV	MAC 360
		by logical operators and parentheses and has a single value, true or false. This value is the result of the calculations specified by the quantities and operators comprising the expression. The logical operators are .NOT., .AND., and .OR. denoting respectively logical negation, logical multiplication and logical addition (also order of precedence). A logical expression may consist of a single logical element. Single elements may be combined through use of the logical operators .AND. and .OR. to form compound expressions, Any logical expression may be enclosed in parentheses and regarded as an element. Any logical expression may be preceded by the operator .NOT.	
3.3.3 Alignment Rules	No	No	No
3.3.4 Precision and Computation Rules	No	DOUBLE PRECISION is allowed but it need not give a full double precision value. Single precision is the normal mode.	All variables are implied double precision and allow 15.9 to 16.8 decimal digits.

Data Element Types, Groups and Operations

3.0	ALGOL 60	FORTRAN IV	MAC 360
3.3.5 Precedence and Sequencing Rules	Evaluated left to right: Exponentiation Multiplication, Division Addition, Subtraction Relational Logical negation Logical conjunction Logical disjunction Logical implication Logical equivalence	Evaluated left to right: Exponentiation Multiplication, Division Addition, Subtraction Relational Logical negation Logical multiply Logical add	In an expression with no parenthesis or within a pair of parentheses, when operations of equal or unequal priority appear, the leftmost operation with the highest priority takes precedence over others and is evaluated first. In an expression involving parentheses, the operation-by-operation evaluation proceeds from left to right, as above, until a parenthesized expression is encountered as one of the operands of the leftmost highest priority operation. Evaluation of the parenthesized expression then takes place by recursive application of this rule and the paragraph above until the operand has been fully evaluated. The interrupted left to right evaluation then resumes. List shows

Data Element Types, Groups and Operations

3.0	ALGOL 60	FORTRAN IV	MAC 360
			priority number in parentheses: Exponentiation (6) Matrix transpose (6) Matrix inverse (6) Scalar-scalar product (5) Scalar-vector or scalar product (5) Scalar-matrix or matrix scalar product (5) Vector-matrix product(5) Matrix-vector product(5) Vector outer product (5) Matrix-matrix product (5) Vector cross product (5) Vector inner (dot) product (3) Scalar-scalar quotient (2) Vector-scalar quotient (2) Matrix-scalar quotient (2) Scalar sum or difference (1) Vector sum or difference (1) Matrix sum or difference (1) Relational Operators (IF's) Boolean Operators (IF's)

Data Element Types, Groups and Operations

3.0	ALGOL 60	FORTRAN IV	MAC 360
3.4 Accessibility of Data			
3.4.1 Hardware Defined	No	No	No
3.4.2 Language Defined	All variable types except strings can be declared and/or operated on.	All variable types can be declared and/or operated on, and take up one machine word (exceptions arise from use of attributes DOUBLE PRECISION and COMPLEX which use two machine words).	All variable types can be declared and/or operated on, and take up two machine words (exception arises from use of address variables which use one machine word).
3.5 Scope of Data	See 1.3.6	See 1.3.6	See 1.3.6

Executable Statements

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.1 Assignment, Exchange and Computation Statements			
4.1.1 Data Element Types			
4.1.1.1.1 Scalar	Variable := expression; No exchange statements.	Variable = expression. No exchange statements.	Variable = expression. No exchange statements.
4.1.1.1.2 Arithmetic	INTEGER variables REAL (Floating Point) variables Integer division is allowed (÷) and requires operands to be of type integer and truncates positive results to next lower integer and negative results to next highest integer. See 1.4.1.1.1	INTEGER variables Floating Point variables DOUBLE PRECISION variables See 1.4.1.1.1	Floating Point variables. Numerical solution of differential equations (derivatives) of form: Dy/Dx = f(x,y) x = dependent variable y = independent variable DIFEQ, DQPHASE statements update variables and control proper sequencing through functions each time executed.

Executable Statements

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.1.1.1.2 Boolean	BOOLEAN variables	LOGICAL variables	No
4.1.1.1.3 Comparison	Yes	No	No
4.1.1.1.4 List or Pointer	No	No	No
4.1.1.1.5 Other	No	Assigned GO TO of form ASSIGN K to i where K = statement label i = integer variable name	Assigned GO TO of form SET K to i where K = address variable i = statement label
4.1.1.2 Non-Scalar	No	Variable = expression No exchange statements.	Variable = expression No exchange statements.
4.1.1.2.1 Arithmetic		COMPLEX attribute	List of variables vector variables matrix variables
4.1.1.2.2 Bit String		No	No
4.1.1.2.3 Text String		No	No

4.0	Executable Statements		
	ALGOL 60	FORTRAN IV	MAC 360
4.1.2 Arrays	No	No	Yes, and arrays can be self-defined - comma separated list of successive elements, enclosed in parentheses.
4.1.3 Hierarchical Structures	No	No	No
4.1.4 Nested Assignment (Factoring)	Multi-variables Multi-expressions	No	No
4.1.5 Conversion Rules for Results	Automatic conversion right to left. See 3.3.2.	Automatic conversion right to left. See 3.3.2.	No
4.2 Textual Data Handling Statements			
4.2.1 Editing	No	No	No
4.2.2 Conversion	No	No	No
4.2.3 Sorting	No	No	No
4.2.4 Comparison	No	No	No

Executable Statements

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.3 Sequence Control and Decision Making Statements			
4.3.1 Unconditional Control Transfer			
4.3.1.1 No Return	By the GO TO command, of form GO TO S where S is the label of some other executable statement. Normal sequencing is from one statement to next.	By the GO TO command, of form GO TO S where S is the label of some executable statement. CONTINUE command simply causes continuation of the normal execution sequence. STOP statements cause return to operating system. Sequencing is from one statement to next.	By the GO TO command, of form GO TO S where S is the label of some executable statement. EXIT, RETURN and TERMINATE statements cause return to operating system. Normal sequencing is from one statement to next.
4.3.1.2 Return	Procedure invocation with supplied parameters and individual procedure name of form: NAME (A_1, A_2,...A_N) and declaration of form: PROCEDURE NAME (A_1,A_2,...A_N) followed by specification statements which give information about procedure parameters followed by statements that make up the procedure body and are enclosed in BEGIN END brackets.	Subroutines are invoked by subroutine calls of form: CALL S (A_1,A_2,...A_N) or CALL S where S is name of subroutine and A_1 are arguments.	Subroutines are invoked by the CALL S, A_1, A_2,... A_N or CALL S where S is name of subroutine and A_i are arguments. DO and DO TO statements of form DO α where α is an executable statement label. DO differs from GO TO in that control returns to statement after DO. The executable statement label is a DO TO statement which establishes the end of the closed routine.

Executable Statements

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.3.2 Conditional Control Transfer			
4.3.2.1 No Return	By the IF, THEN, ELSE command; three types as follows: IF B THEN U IF B THEN U ELSE S IF B THEN F or IF B_1 THEN S_1 ELSE IF B_2 THEN S_2, etc. where: B = Boolean expression U = Unconditional statement S = Any statement F = FOR statement The switch declaration is of form SWITCH NAME := A_1, A_2,...A_N where NAME identifies switch and is followed by switch list which consists of an array of labels separated by commas. Switches are invoked by GO TO NAME [arithmetic expression];	Arithmetic IF imperative of form IF (E) K_1, K_2, K_3 where E = arithmetic expression K = statement labels Control is transferred to K_1, K_2, K_3 as E is less than zero (K_1), zero (K_2) and greater than zero (K_3). Logical IF of form IF (E) S where E is logical expression and S is any executable statement except DO or another IF. Computed GO TO of form GO TO (K_1,K_2,...K_N), I where K_i is statement label and I is an integer. Assigned GO TO of form GO TO I, (K_1,K_2,...K_N) where I is a variable of the type integer and K_i are statement labels.	By the IF, OTHERWISE commands of form: IF A_i, B_j IF A_i, B_j, OTHERWISE B_k where A is an IF conditional, B is any executable statement. Binary logical operators (AND,OR) can be part of IF conditionals. Computed GO TO of form: GO TO (X),A_1,A_2,...A_N where X = scalar arithmetic expression and A_i statement labels associated with some executable statements. Assigned goto of form SET β TO α and invoked by GO TO β.
4.3.2.2 Return	No	No	Computed DO of form: DO (X), A_1,A_2,...A_N where X and A_i are same as computed GO TO.

Executable Statements

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.3.3 Loop or Index Control			
4.3.3.1 Loop Designation	Concise Loop Control with the FOR imperative. This consists of a FOR list, followed by DO and followed by a statement. FOR list consists of a single parameter which is to be varied followed by list over which it is being varied and form is: FOR I := L_1,L_2,...L_N statement where L_i = single valued arithmetic expression.	Concise Loop Control with the DO imperative of the form DO L I = N_1, N_2, N_3 where L is statement label and N_i are loop parameters. If N_3 is omitted it is assumed to be an integer one.	Concise Loop Control with the DO TO α for V = F; where α is a DO TO object, V is scalar variable and F = range expression.
4.3.3.2 Range of Loops	Statement label following DO imperative.	Sequence of statements physically following the DO through and including the statement with label N. The CONTINUE statement can be used as a target point for transfers, particularly as the last statement in the range of the DO statement.	The statement specified by α.

Executable Statements

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.3.3.3 Iteration Control Mechanism	These items are of the form: A STEP B UNTIL C DO statement where I = index A = initial index value B = increment C = final index value where all three can be arithmetic expressions; loop terminates when A > C (+B) and/or A < C (-B). An alternative form is: E WHILE F DO statement where E = arithmetic expression F = Boolean expression which results in repetition of E until F is false.	There is a single parameter I which is varied by assigning it the first value N_1 and incrementing by N_3 after Control reaches statement N. The loop is considered finished only when I exceeds the value of N_2. The terminal statement designated by L cannot be GOTO, IF, RETURN, STOP, PAUSE or DO. DO statements can be nested.	There is a single parameter V which is varied by assigning it the first value of the range expression and incrementing by the second value. The incremented value is compared with the third value after the loop is processed but before incrementing; the variable is checked for $\geq$ and if met, loop is terminated. The effect is to cause repeated execution of some statement identified by α as the variable V takes on its range of values.
4.3.4 Real Time Control			
4.3.4.1 Multi-tasking	No	No	No

4.0	Executable Statements		
	ALGOL 60	FORTRAN IV	MAC 360
4.3.4.2 Scheduling and Dispatching	No	No	No
4.3.4.3 Storage Allocation			
4.3.4.3.1 Static	No	No	No
4.3.4.3.2 Dynamic	No	No	No
4.3.4.4 Access Restrictions			
4.3.4.4.1 Memory	No	No	No
4.3.4.4.2 Registers	No	No	No
4.3.4.4.3 Interrupts	No	No	No
4.3.4.5 Interrupt Handing	No	No	No
4.3.5 Error Condition and Program Checking	No	No	No

Executable Statements

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.4 Symbolic Data Handling			
4.4.1 Algebraic Expression Manipulation	No	No	No
4.4.2 List Handling Statements	No	No	No
4.4.3 String Handling Statements	No	No	No
4.4.4 Pattern Handling Statements	No	No	No

Executable Statements

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.5 Interaction With Operating System and/or Environment			
4.5.1 Input/ Output Statements	There is a proposed ISO standard - two types: one a very small subset of the other; as follows:		
4.5.1.1 File Initialization/ Processing/ Termination	INSYMBOL, OUTSYMBOL, LENGTH, INREAL, OUTREAL, INARRAY, OUTARRAY.	The formatted (fixed) form for I/O is as follows: READ/WRITE (U,F) K where U = logical file name F = FORMAT statement label K = list of variables The unformatted (free) form for I/O is as follows: READ/WRITE (U) K I/O devices are specified by integer constants or variables used in the READ/ WRITE imperatives. There is a separate FORMAT statement that provides data format descriptions.	READ statement of form: READ α where α = input list and provides reading of fixed or comma delimited fields from input cards. Or RDFLD (β) where β = scalar arithmetic expression and provides reading of free form data fields from input cards. Devices are usually specified in I/O statement. There are FORMAT and LONG FORMAT statements that provide data format descriptions.

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.5.1.2 File Positioning and Handling	No	BACKSPACE U END FILE U REWIND U	The following are I/O commands available: PRINT HDG, α RDPTB α PRINT MSG, α RDPTG α PRINT FORMAT N, α PRINT α PUNCH HDG, α PUNCH MSG, α PUNCH FORMAT N, α PUNCH α TYPE HDG, α TYPE MSG, α TYPE FORMAT N, α TYPE α where α = output list N = label of FORMAT statement SP0 SP1 SP2 Vertical SP3 Format Control SP4 SKIP BLANK - Horizontal format control MAC Datafile I/O uses; SET FILE READ α SET FILE WRITE α and FILE READ α FILE WRITE α where α = scalar arithmetic expression

Executable Statements

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.5.2 Library Reference Statements	No, available through procedure call (standard functions).	Intrinsic and basic external functions. Others through subroutine mechanism.	No, all functions.
4.5.3 Debugging Statements	No	No	See 6.5
4.5.4 Storage and Segmentation Allocation Statements			
4.5.4.1 Static	No, implementation specific and rather elaborate.	No	No, however one can control storage use with CALL (NONMAC, RESIDENT) statement.
4.5.4.2 Dynamic	No, implementation specific and rather elaborate to handle arrays with dynamic bounds and recursive procedures.	No	No, same with CALL (NONMAC) statement.
4.5.5 Operating System and Machine Dependent Statements	No	STOP, PAUSE imperatives with option of 5 or less octal numerals specifying transfer address.	TERMINATE statement.

Executable Statements

4.0	ALGOL 60	FORTRAN IV	MAC 360
4.5.6 Others	No	No	Program collection: in preparation for execution of a MAC program it and all supporting MAC programs are collected from the MAC program file and stored in a working program file (includes NONMAC support programs).

Non-Executable Statements

5.0	ALGOL 60	FORTRAN IV	MAC 360
5.1 Data Declarations			
5.1.1 Data Element Type Declarations			
5.1.1.1 Constants			
5.1.1.1.1 Arithmetic			
5.1.1.1.1.1 Hexadecimal	No	No	No
5.1.1.1.1.2 Decimal	Integer Floating Point	Integer Floating Point (Single, Double Precision) Complex (Floating Point)	Floating Point (Double Precision)
5.1.1.1.1.3 Octal	No	No	No
5.1.1.1.1.4 Binary	No	No	No

Non-Executable Statements

5.0	ALGOL 60	FORTRAN IV	MAC 360
5.1.1.1.2 Boolean	TRUE FALSE	.TRUE. .FALSE.	No
5.1.1.1.3 Textual	The quote marks are used to delimit literal strings. The only place where literal strings appear are as parameters of procedure calls.	Text string of form n H C_1 $C_2 \ldots C_N$ where: n = number of characters H = Hollerith code	No
5.1.1.1.4 Other	No	No	No
5.1.1.2 Variables	The first appearance of an identifier in a program must be in the form of an implicit declaration. Labels need not be declared. Loop variables are contextually declared. Declarations appear in block heads.	Implicit declarations formed from first letter of identifier. Integer variables have name beginning with letters I through and including N. Any other letter implies a floating point variable. Explicit declarations override this convention. Labels are contextual declarations because they are determined by the context in which it appears. Arrays are explicitly declared via the DIMENSION statement.	Declarations are self defining for the most part. Index variables are explicitly declared. Labels are contextually declared. Arrays larger than 3x3 are explicitly declared with the DIMENSION statement.

Non-Executable Statements

5.0	ALGOL 60	FORTRAN IV	MAC 360
5.1.1.2.1 Arithmetic	REAL declares the variable to be a real number (floating point). INTEGER declares the variable to be an integer.	REAL declares the variable to be a real number (floating point). INTEGER declares the variable to be an integer. DOUBLE PRECISION declares the variable to be real but allowing approximately twice as many significant digits as the single precision REAL declaration. COMPLEX declares the variable to be imaginary and as such is non-scalar and two real numbers.	Implied real double precision variables.
5.1.1.2.2 Boolean	BOOLEAN declares the variable to be a boolean and its value will be either true or false.	LOGICAL declares the variable to be a boolean and its value will be either true or false.	No
5.1.1.2.3 Textual	No	No, although assignment statements can be used. At one time there was a HOLLERITH data type but it was dropped.	No

Non-Executable Statements

5.0	ALGOL 60	FORTRAN IV	MAC 360
5.1.1.2.4 Other	OWN variables have permanent storage reserved for them at translation time. This prevents the erasing of values of variables on exit from a block.	No	No
5.1.1.3 Presetting of Declarations	No	DATA declaration of form DATA K_1/d_1, K_2/d_2/,..., K_N/d_N/ where K_i is a list of names of data elements and arrays, and d_i is a list of values of these names. Values may be numeric and alphanumeric constants. The notation n*d indicates n repetitions of the value d.	No
5.1.1.4 Nesting (Factoring) of Declarations	Yes	Yes	No
5.1.1.5 Default Options	No	No	No
5.1.1.6 Numbering Conventions	No	No	No

Non-Executable Statements

5.0	ALGOL 60	FORTRAN IV	MAC 360
5.1.2 Group Type Declarat Declarations			
5.1.2.1 Array Declarations	The symbol ARRAY declares the item an array and it is followed by the array name followed by the number of dimensions of the array and includes the upper and lower bounds (encoded by [ ] pair). There can be any number of dimensions and subscripts may be arbitrarily complex arithmetic expressions of integer type. Array declarations appear in block heads along with other declarations. The amount of space required for an array cannot be predicted during translation or loading but is dynamically allocated during execution.	The maximum dimensions of an array are specified by the DIMENSION statement. This serves to reserve space in memory. The space allocated during translation is fixed and cannot be adjusted during execution. The form is DIMENSION followed by array name followed by number of dimensions and number of elements per dimension (enclosed in ( ) pair). The dimension specifications are integer constants. If integer variables are used this provides for variable dimensions. There can be no more than three dimensions. Arrays may be declared in the COMMON or type declarations.	The DIMENSION statement has the form: DIMENSION Y1,Y2,Y3,..., Yn where Y1 to Yn are array variable dimension-defining expressions. Any i is either an explicit expression or an implicit expression. An explicit expression specifies the dimensions of a particular array variable and has one of the forms: (α, n x m) (α, n) An implicit expression specifies the dimensions of all array variables in the program not mentioned in some explicit dimension expression. A program

5.0	ALGOL 60	FORTRAN IV	MAC 360
			must contain no more than one such implicit dimension definition. An implicit expression has one of the forms: n x m n where: α is an array variable name; n and m are pure integers in the interval [1,255]; and n by itself implies nXn. If a program contains no DIMENSION statement or no implicit-dimension specification, an implicit dimension of 3 is assumed by the compiler. The RESERVE statement has the form: RESERVE Y1,Y2,Y3,...,Yn where Y1 to Yn are variable names. An index expression associated with any array variable name in the list must be a simple integer literal. The sole function of the RESERVE statement is to establish the sizes of certain local arrays.

Non-Executable Statements

5.0	ALGOL 60	FORTRAN IV	MAC 360
5.1.2.2 Hierarchical Structures	No	No	No
5.1.2.3 Procedure, Function, Subroutine Declarations	Yes, See 4.3.1.2	Yes, See 4.3.1.2	Yes, See 4.3.1.2
5.1.2.4 Presetting	No	See 5.1.1.3. In addition, the BLOCK DATA statement declares the program that follows to be a data specification subprogram. The subprogram may contain only the declarative statements associated with data being defined.	No
5.1.2.5 Nesting (Factoring) of Declarations	Yes	Yes	Yes
5.1.2.6 Default Options	No	No	Assumes 3x3 arrays.
5.1.2.7 Numbering Conventions	See 1.3.5.4	See 1.3.5.4	See 1.3.5.4

Non-Executable Statements

5.0	ALGOL 60	FORTRAN IV	MAC 360
5.1.3 Interaction with Operating System and/or Environment			
5.1.3.1 File Declarations	No	No	No
5.1.3.2 Storage and Segmentation Declarations	No, although ARRAY and blocks convey this type of information to compiler.	The EQUIVALENCE statement allows more than one identifier to represent the same quantity. The form is: EQUIVALENCE $(A_1,A_2,\ldots)$, $(A_N, A_{N+1},\ldots),\ldots$ where A is a reference or an arbitrary number of groups of variables that are set to be equivalent to each other. They can also be used to equivalence arrays to each other.	No
5.1.3.3 Hardware Declarations	No	No	No

Non-Executable Statements

5.0	ALGOL 60	FORTRAN IV	MAC 360
5.1.3.4 Format Declarations	Number, string, non-format title, alignment marks; standard (large set).	All formatted input or output requires the use of a data format specifying the external format of the data and the type of conversion to be used. The FORMAT statement provides this and form is: FORMAT(S_1,S_2,...,S_N) where S = data field specification of following types: Numerical Fields (P,E,F,G,I) Scale Factors G-Fields Logical Fields (L) Alphanumeric Fields (A) Blank or Skip Fields (X) Complex Fields Multiple Record Formats	All formatted output requires the use of the data format specifying the external format of the data. The FORMAT, LONG FORMAT statement provides this and form is: FORMAT n LONG FORMAT n where n, the format reference label is a simple integer (0-99999) and followed by 1,2, or 3 cards of text, interspersed with data format and scaling indications. Five formats and scaling types are allowed. The LONG FORMAT statement allows more columns per line. The other output commands are automatically formatted by the compiler.
5.2 Compiler Directives			
5.2.1 Optimization			

Non-Executable Statements

5.0	ALGOL 60	FORTRAN IV	MAC 360
5.2.1.1 Space	No	No	No
5.2.1.2 Time	No	No	No
5.2.1.2 Time	No	No	No
5.2.2 Debugging Aids	No	No	See 6.5.2
5.2.3 Documentation	No	No	No
5.2.4 Documentation Control	No	No	No
5.2.5 Storage Control	No	No	NEW BLOCK which instructs compiler to start a new block with the next encountered MAC statement.

Non-Executable Statements

5.0	ALGOL 60	FORTRAN IV	MAC 360
5.2.6 Compilation Control	No	No	REMAC implies that the source deck to be compiled is in the MAC symbolic program storage file. The REMAC card is followed by update cards. PRODMAC, PRODREMAC removes trace linkage from object program.
5.2.7 Others	No	No	No

Structure of Language and Compiler Interaction

6.0	ALGOL 60	FORTRAN IV	MAC 360
6.1 Self-Modification of Programs	No	No	No
6.2 Bootstrapping	No, there are no character handling facilities, but specific algorithms have been written in ALGOL with the necessary features.	No	No
6.3 Extensible	No, but a system has been proposed.	No	No
6.4 Subsets or Dialects	No	It is possible that a FORTRAN program be handled by a basic FORTRAN compiler.	No
6.5 Debugging, Parametric Programming Facilities			
6.5.1 Compilation Time	No	No	No
6.5.2 Object Time	No	No	START TRACE, STOP TRACE, TRACE, TRACE ONE provide trace of statements in object program. The trace mechanism can easily be controlled to trace what the user desires.

Structure of Language and Compiler Interaction

6.0	ALGOL 60	FORTRAN IV	MAC 360
6.6 Effect of Language Design On Implementation Efficiency, Remarks.	Problems arise from following: integer labels own variables recursion in procedures unknown array sizes identifier size number of variables Various implementations treated these problems differently.	Good object time efficiency and somewhat poor compilation time. Many compilers are available.	Object time efficiency is not good. Compile time is fast. Run time storage paging automatic with program blocks and datafile blocks. The language syntax is not clear. There are things permitted in some contexts but not others which tends to make compiler more complicated. The three line format complicates the syntactic and lexical analysis.

Library Procedures, Functions

	ALGOL	FORTRAN	MAC
1. Conversion Procedures			
A. Input			
1) Convert EBCDIC to ASCII			
2) Convert integer to floating point		✓	
3) EBCDIC hexadecimal to unsigned binary			
4) EBCDIC integer to signed binary			
5) EBCDIC octal to unsigned binary			
B. Output			
1) Convert ASCII to EBCDIC			
2) Convert signed fixed-point to num to EBCDIC			
3) Convert floating-point number to integer		✓	
4) Convert integer to EBCDIC hexadecimal			
5) Convert integer to EBCDIC decimal			
6) Convert single precision to double precision		✓	
2. Mathematical Procedures			
A. Trigonometric Functions (Scalar-Scalar)			
1) Sine			
real	✓	✓	✓
double precision		✓	
complex		✓	
2) Cosine			
real	✓		
double precision			
complex			

	ALGOL	FORTRAN	MAC
3) Tangent			✓
4) Secant			✓
5) Cosecant			✓
6) Cotangent			✓
B. Inverse Functions (Scalar-Scalar)			
1) Arc-sine			✓
2) Arc-cosine			✓
3) Arc-tangent			
real		✓	
double		✓	
4) Arc-secant			
5) Arc-cosecant			
6) Arc-cotangent			
C. Hyperbolic Functions (Scalar-Scalar)			
1) Hyperbolic Sine			✓
2) Hyperbolic Cosine			✓
3) Hyperbolic Tangent		✓	✓
4) Hyperbolic Secant			✓
5) Hyperbolic Cosecant			✓
6) Hyperbolic Cotangent			✓
D. Inverse Hyperbolic Functions (Scalar-Scalar)			
1) Inverse Hyperbolic Sine			✓
2) Inverse Hyperbolic Cosine			✓
3) Inverse Hyperbolic Tangent			✓
4) Inverse Hyperbolic Secant			
5) Inverse Hyperbolic Cosecant			
6) Inverse Hyperbolic Cotangent			

	ALGOL	FORTRAN	MAC
E. General (Scalar-Scalar)			
1) Natural Logarithm			
real	✓	✓	✓
double precision		✓	
complex		✓	
2) Common Logarithm			
real		✓	✓
double precision		✓	
3) Compute e^x			
real	✓	✓	✓
double precision		✓	
complex		✓	
4) Square Root			
real		✓	✓
double precision		✓	
complex		✓	
5) Integer Divide			
6) Random Number (normal)			✓
7) Random Number (rectangular)			✓
8) Absolute Value			
real	✓	✓	✓
double precision		✓	✓
integer		✓	✓
9) Entier	✓		
F. Array (Vector, Matrix-Scalar, Vector-Vector, Matrix-Matrix)			
1) Absolute Value			✓
2) Determinant			✓
3) Trace			✓
4) Unit			✓
5) Adjoint			✓
6) Inverse			✓
7) Transpose			✓

	ALGOL	FORTRAN	MAC
8) IDMATRIX			✓
9) MCDET			✓
G. Special			
1) Summation			✓
2) Product			✓
3) Satisfy			✓
4) Maximum		✓	✓
5) Minimum		✓	✓
6) DQPHASE			✓
7) DIGITB, DIGITG			✓
8) RDFLD, RDPTB, RDTPG			✓
9) SIGF			✓
10) SGN			✓
11) SIGN	✓		✓
12) ROUND			✓
13) TIME_OF_DAY			✓
14) Truncation			
real		✓	✓
double precision		✓	
15) Remaindering (Double Precision)		✓	
16) Transfer of Sign		✓	
17) Positive Difference		✓	
18) Real Part of Complex Argument		✓	
19) Imaginary Part of Complex Argument		✓	
20) Express Two Arguments (Real) in Complex Form		✓	

	ALGOL	FORTRAN	MAC
21) Conjugate of Complex Argument		✓	
22) Modulus (Complex)		✓	
23) Most Significant Part of Double Precision Argument		✓	