ESD-TR-68-455

(Final Report) ESLFE

APPROACH FOR CHANGE
JOVIAL EVALUATION PROJECT

William M. O'Brien

December 1968

COMMAND SYSTEMS DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

(Prepared under Contract No. F19628-68-C-0301 by Data Dynamics, Inc., 9800 S. Sepulveda Boulevard, Los Angeles, California 90045.)

LEGAL NOTICE

When U. S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

ESD-TR-68-455

(Final Report)

APPROACH FOR CHANGE
JOVIAL EVALUATION PROJECT

William M. O'Brien

December 1968

COMMAND SYSTEMS DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

(Prepared under Contract No. F19628-68-C-0301 by Data Dynamics, Inc., 9800 S. Sepulveda Boulevard, Los Angeles, California 90045.)

FOREWORD

The Approach for Change presented herein is one of the primary tools used to evaluate the JOVIAL (J3) computer programming language as specitied in AFM 100-24. The use of this manual is limited in scope due to its primary purpose, that of providing criteria for the evaluation of the language.

The Approach for Change was produced primarily by William O'Brien of Data Dynamics as part of Project 6917, Task 04, under contract F19628-68-C-0301 for Electronic Systems Division, Air Force Systems Command. The project monitor was Capt. Martin J. Richter.

This questionnaire has been reviewed and approved by Hq. USAF and is assigned the reports control symbol HAF-D-86-(OT).

Martin J. Richter

MARTIN J. RICHTER, Capt., USAF
Project Monitor

William F. Heisler

WILLIAM F. HEISLER, Col., USAF
Chief, Command Systems Division
Directorate of Planning & Technology

ABSTRACT

The Approach for Change document was produced to establish the criteria for evaluating features of the JOVIAL (J3) computer programming language as specified in AFM 100-24. The document contains arguments for retaining, deleting, or modifying JOVIAL features including a methodology for resolving the arguments.

TABLE OF CONTENTS

		Page
FOREWORD		ii
ABSTRACT		iii
SECTION I	Introduction	1
SECTION II	Languages, Standardization, and Function Modules	2
SECTION III	Retentions, Deletions, Modifications, and Resolutions	4
	3.1 Retention and Deletion of Function Modules	4
	3.2 Modification of Function Modules	4
	3.3 Retention and Deletion of (Non-Nucleus) Features	5
	3.4 Modification of Features	6
SECTION IV	Other Considerations	7
	4.1 Determining Features of Universal Merit	7
	4.2 Compatibility Considerations	7
APPENDIX	List of Features/Functional Modules with Arguments for Retention, Deletion, Modification, and Resolution Criteria	8

SECTION I

Introduction

This document constitutes an approach for changing J3 (AFM 100-24) on the basis of information obtained in the course of the JOVIAL Evaluation Project, (Contract No. F19628-68-C-0110). The major part of the document is a list of J3 features and function modules accompanied by arguments for retaining, deleting, and/or modifying them together with schemes for "resolving" these arguments. (This list is contained in the appendix.)

Section 2.0 describes some pertinent facts about languages, standardizing, and the notion of function modules employed. Section 3.0 shows the basic slant of the arguments made and describes resolution schemes. Section 4.0 discusses some peripheral considerations. The concepts of this document may be modified on the basis of insights gained at interview sessions.

SECTION II

Languages, Standardization, and Function Modules

Very briefly, a language is a notational system with a community of users. The structure of the language "sentences" and the meanings attached to them reflect both

- a set of problems shared by members of the community, and
- a set of tools which make the problems tractable.

To standardize a language means to determine the problems faced by all, or a majority, of the members of the community and to obtain a consensus as to the tools to be employed in solving the common problems. Clearly, a language standard should not support tools designed for coping with problems peculiar to a small subcommunity nor should the standard support a variety of sets of tools designed to handle the same set of common problems. The importance of standardizing a language lies in the fact that more efficient communication of problems and problem solution will, in general, ensue. Standardization creates a common frame of reference for the members of the community of users.

The language which is the object of this study is, of course, the J3 dialect of JOVIAL specified in AFM 100-24; the community of users consists of all those applications personnel who are employing JOVIAL in computerized command and control systems. The problem solving tools are the J3 language features themselves and, equally important, sets of features, which in this document are referred to as function modules. It is felt that evaluation of individual features in an isolated fashion is insufficient because many related features, or tools, are generally brought to bear in concert to solve command and control problems. Thus, the significance of fixed point items, say, can be meaningfully understood only in the context of the other numeric items (integer and floating point), since numeric items are functionally connected in the sense that all deal in one way or another with fractional point. Similarly, the significance of numeric items can be better understood when examined in the context of arithmetic operations. "Numeric Items and Arithmetic Operations" constitute a subsystem of individual J3 features which address the general problem of representing numeric entities and doing arithmetic with them. This document, then, relies heavily on the concept of problem solving subsystems, or function modules, to clarify command and control programming processes and to aid in the evaluation of individual features.

At this point it is illuminating to rephrase the statement about the importance of standardizing languages in terms of J3. Standardization of J3 is important for two reasons:

- it allows the transfer of programs from one computer system to the other, and
- it allows the transfer of programmers from one command and control application to the other.

It is felt that the first reason --- transferring programs between computers --- is possibly of considerably less importance than the second. The assumption is that command and

control programs tend, on the whole, to be much more application specific than routines written, say, in FORTRAN, making transferral either meaningless or highly impractical. This assumption needs to be validated, of course, in the light of JAQ and interview responses.

The second reason --- transferring programmers between applications --- is considered extremely important, and the value judgments brought to bear in this document reflect this notion. Great emphasis is placed on the importance of programmer convenience and usability.

SECTION III

Retentions, Deletions, Modifications, and Resolutions

Arguments for retention, deletion, and/or modification of features and function modules are presented in the appendix on an individual basis for each feature and function module in J3. Certain features and function modules have been deemed so important that retention without question is recommended --- these features and function modules are denoted in the appendix as Nucleus language facilities. For most non-nucleus features arguments for both retention and deletion have been given. Obviously, some mechanism for determining whether to delete or retain must be provided in these cases. This mechanism is called the resolution and provides a means for deciding whether to delete or retain on the basis of usage rates (see 3.3).

This section describes in general terms the kinds of arguments for retention, deletion, and modification which are made in the appendix.

3.1 Retention and Deletion of Function Modules

Most functional modules are considered nucleus and will not be deleted. A recommendation will be made to delete a function module which is not nucleus if all features in the module are recommended for deletion. Otherwise, the function module will be retained. Retained function modules may be modified by the deletion of current features or the additions of new ones, as described below.

3.2 Modification of Function Modules

Typical arguments for modification are:

- By altering one or more features and/or adding others it is possible to obtain uniqueness of function in the features comprising the module. Uniqueness of function means that a feature performs precisely one function and that that function is clearly defined and separated from non-related functions. This property is regarded as desirable because it makes clear the relationship between problem and tool and thereby aids the programmer. (OVERLAY is an example of a feature which performs several functions; packing has to do with allocation, not with tables.)
- By altering one or more features and/or adding others it is possible to obtain more power in the language. (Example: Addition of a non-recurrent COBOL-record-like data structure might provide a worthwhile expansion of utility.)

Note that the modifications to function modules proposed in the appendix will be used to provide a basis of discussion at interviews. Recommendations to implement modifications will be made on the basis of the degree of interest shown at interviews.

3.3 Retention and Deletion of (Non-Nucleus) Features

Typical arguments for retention are:

- the feature provides a convenience to the programmer in the function it performs
- retention helps to complete a subsystem. Example: AND, OR, NOT are not required to write J3 programs, but if they are retained, a relatively complete logical algebra subsystem is available to the programmer. Such a subsystem is convenient to use in solving decision making programs and is well known (or easily explained) to programmers. If the logical algebra subsystem is retained, therefore, decision making processes in J3 can be described under one familiar and comprehensive conceptual roof, making it easier to transfer a program design to actual code.

Typical arguments for deletion are:

- the function performed by the feature can be performed by other (nucleus) features. Carrying non-essential features means an expense in compiler construction and maintenance, as well as extra compilation time.
- non-essential features require more learning on the part of the programmer and limited use may result in a large number of errors.
- there is some unusual difficulty in implementing this feature in a compiler. (This case arises very infrequently and is always noted in the appendix. When no special difficulty is foreseen, no mention is made. In general, no particular problems in maintaining a feature in a compiler have been contemplated.)

The resolution mechanism used is based in most instances on usage rates. The usage rate of a feature for any given application is computed by taking the ratio of the number of occurrences of a feature (numerator) to the number of occurrences of one or more other features (denominator). The denominator is ordinarily developed on the basis of those features which must be used before the feature in question may be used. Thus, the denominator developed for the usage rate ratio of parallel tables would be the number of occurrences of tables, since parallel/serial has no meaning outside the context of table. The typical scheme for determining the resolution is:

- compute a usage rate for the given feature for each application
- retain the feature if the usage rates for the majority of applications is not less than some threshhold usage rate. A specific threshhold is given for the resolution of each feature and is based on our weighing the arguments for deletion against those for retention. A high threshhold would indicate we lean toward the deletion arguments. Please note that threshhold usage rates are subject to change based on interview responses.

Note that this technique corresponds to the consensus (majority) of users in the community of users.

In the appendix, resolutions typically appear as: "Retain if U GE 0.05 where U = (ratio)".

"U GE 0.05" is to be read "majority of usage rates greater than or equal to the threshhold value of 0.05". "Where U = (ratio)" shows the formula for computing the usage rate for that feature for each application.

Sometimes the denominator will be shown in the form, Numeric + Status + Boolean + Literal; this is to be read as "number of occurrences on Numeric items + number of occurrences of Status items +. . .etc."

3.4 Modification of Features

Modification of features will be recommended for much the same reasons as modifications of function modules.

SECTION IV

Other Considerations

4.1 Determining Features of Universal Merit

By definition, those features which are nucleus are features of universal merit. The standard deviation of usage rates of a feature may be used as a pragmatic measure of the "degree of universality" of the feature. More precisely, one would:

(a) Compute the mean, U_i, over all applications for each feature, f_i.
(b) Compute the standard deviation, S_i, for each U_i.
(c) Rank the features from lowest S_i to highest. Standard deviation represents spread from the mean; the greater the spread, the greater the disagreement among users on the universality of the feature. Therefore, those features at the top of the list would be regarded as having universal merit. Those at the bottom would vary in merit from application to application.

4.2 Compatibility Considerations

Compatibility of applications dialects with the Standard (AFM 100-24) will have to be treated on an individual basis. Degree of compliance of an application dialect with the standard will be determined from responses at interviews and in the JAQ. Particular care will be given to the role played by the operating system.

APPENDIX

List of Features/Function Modules

with Arguments for Retention, Deletion

Modification and Resolution Criteria

A1. Function Module: Data (Nucleus)

Function: To provide for the representation of entities associated with the processes simulated by programs.

Retention: Fundamental.

Resolution: Retain.

A1.1 Function Module: Item Data (Nucleus)

Function: To provide for the representation of a variety of elementary entities, namely: numeric quantities, character strings, statuses of processes being simulated, and logical data.

Retention: It is the point of view of J3 that command and control problems are tractable in terms of the item types provided. There is no apparent reason to suspect otherwise at this time.

Modification: Status and logical items may be represented in terms of numeric items and may be deleted as described below.

Resolution: Retain the capability of representing at least numeric and character items.

Note: Numeric and Boolean items are discussed elsewhere. (The Dual "item" is regarded as an aggregate.)

A1.1.1 Function Module: Character Data (Nucleus)

Function: Provides for the manipulation of Character Data. Characters are letters, digits, or special characters (+, $, etc.).

Retention: It is felt that this facility is a requirement for command and control data processing.

Modification: There are two kinds of character data in J3 --- Hollerith and STC. Hollerith implies a specific character set while STC implies both a specific character set and a specific character encoding scheme (representation of a character in terms of bit patterns). Note that:

(1) character set and encoding schemes are two separate aspects of character data which are mixed together in J3 character representation. (Contradiction of "uniqueness of purpose" principle).
(2) The notion of character set is machine independent while that of encoding scheme is not.

(3) It is conceivable that the occasion might arise in which it would be desirable to specify more than one encoding scheme for the same character set.

For these reasons, it may be worthwhile to separate the character set and encoding scheme attributes of character data declarations.

Resolution: Retain the Character Data Facility. (Adoption of the modification is rcommended if STC is retained and/or if the ability to maintain other encoding schemes is requested by users at the interviews.)

A1.1.1.1 Feature: Item Type = "Hollerith" (Nucleus)

Function: Permits formation of character strings from a set of characters consisting of at least the J3 signs together with other characters peculiar to the implementor's computer system.

Retention: Required.

Modification: It is possible, and sometimes more convenient to represent Hollerith (and STC) constants using quotation marks. 'XYZ' instead of 3H(XYZ). (A quote may be represented by two quote marks.)

Resolution: Retain. (Recommendation to implement modification will be made based on interview responses.)

A1.1.1.2 Feature: Item Type = "Standard Transmission Code"

Function: Permits formation of character strings from a set of STC characters. STC characters are encoded in a predefined manner.

Deletion: STC characters are a subset of the Hollerith character set. Therefore, STC character strings can be stored internally as Hollerith items, using subroutines to convert Hollerith to and from STC for input/output.

Retention: The importance of STC is that it constitutes a standard set of characters with a standard encoding scheme which may be transmitted between members of a large family of standardized character handling devices. If the need is great in command and control systems for processing and transmitting STC character strings, the Standard should support some formal mechanism to do so for the sake of programmer convenience.

Resolution: Retain if U GE .10 where

$$U = \frac{STC}{STC + HOL}$$

A1.1.1.3 Feature: Character Data Size Attribute (Nucleus)

Function: Permits specification of number of characters in the character string.

Retain: This feature is required.

Resolution: Retain.

A1.1.2 Feature: Item Type = "Status"

Function: Provides the ability to represent the various (finite number of) states that a process being simulated may take.

Deletion:

(1) Any other item type can be used to perform this function --- use of status items is equivalent to setting and testing "flags".
(2) Status constants are context sensitive because the same status constant may be associated with more than one status variable. (Context sensitivity means extra work for a compiler, since it must determine the contexts in which syntactic constructs appears in order to properly translate.)

Retention: Status items are very convenient to the programmer since the status constant is mnemonic. Thus, "IF ORBIT'MODE EQ V(REENTRY) $" is more readable than "IF ORBIT'MODE EQ 3$".

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Status}}{\text{Numeric} + \text{Boolean} + \text{Literal}}$$

A1.1.3 Feature: Explicit Status Size Attribute

Function: In effect, this attribute has as its value the precision, or maximum number of binary digits to be employed in registering the status constant.

Retention: Permits the programmer to reserve a larger container for the status item than is required to register all the values of the status item. (This is perhaps useful in table packing.)

Deletion: Overlaying the Status Item with an unsigned integer of the desired length will perform the same function.

Resolution: Retain if U GE .10 where

$$U = \frac{\text{Status with Size Specified}}{\text{Status}}$$

A1.2 Function Module: Data Aggregations (Nucleus)

Function: Provides the formal mechanisms for grouping item data together into aggregates. In J3, aggregate structures which may be declared are arrays, tables, strings, and files. (Dual "items" are aggregate structures also and are discussed in A5. Files are discussed in A10.)

Retention:

(1) Formal aggregation facilities are, strictly speaking, not required in the language. It is possible to obtain the same effect by using (individually named) elementary items and embedding the desired aggregation properties in the procedural code. However, not having formal aggregation facilities places enormous burdens on programmers.
(2) The importance of aggregating item data into structures is that the structures bear more information than the component elementary item data comprising them. This "information of structure" is frequently as important in problem parameterization as the information carried by "free standing" item data. Information of structure can include
    - the common properties by which elements of the groupment are related,
    - schemes of recurrence of elementary structures (tables and arrays),
    - significance of position of a datum in a recurrent structure, and
    - logical subordination of one elementary substructure by another.

Modification: J3 aggregates are slanted toward recurrent structures. It is frequently the case that data structures are required which are not recurrent in nature at all. The COBOL record structure is a case in point.

Resolution: Retain the functional capability for providing formal aggregation facilities. (Recommendation for implementation of some scheme for non-recurrent aggregation will be made depending on user requirements voiced at interviews.)

A1.2.1 Feature: Array (Nucleus)

Function: To provide for recurrent structures of single items.

Retention: It is felt that arrays are the most basic kind of recurrent aggregate structures.

Resolution: Retain.

A1.2.2 Feature: Table

Function: To provide for recurrent structures to sets of (possibly) different types of items. (Special case: set consists of one item.)

Retention: It is our notion of tables as they exist apart from computer programs that they are structures of recurrent sets of items. Table structuring and manipulation seems a useful feature to have in command and control programming.

Deletion: The function performed by tables can be performed by using a set of (one dimensional) arrays.

Modification: The formal mechanisms for declaring tables in J3 mixes together the concept of table as a recurrent structure with concepts of storage allocation (packed table and parallel/serial attributes discussed elsewhere). Whereas it is true that controlling storage allocation is of particular importance in recurrent structures, it is well to point out that the storage allocation problem is not inherently connected with tables.

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Tables}}{\text{Tables} + \text{Arrays}}$$

A1.2.3 Feature: Variability Attributes of Tables and NENT

Function: To provide the means for automatically updating the "logical" table length of a table in which the number of entries may change throughout the execution of the program. Functional modifier NENT is set and used procedurely to maintain table length.

Retention: Dynamic variability of the number of entries in a table is a property of command and control programming (e.g., number of aircraft in a sector, number of radar stations capable of acquiring a satellite in a given revolution, etc.) This feature provides a user convenience since it automatically allocates and "names" an unsigned integer variable (the NENT) for the purpose of representing during execution time the number of entries in the table.

Deletion: The function performed by this feature can be performed by the programmer if he declared his own "NENT".

Resolution: Retain if U GE .10 where

$$U = \frac{\text{Rigid Tables}}{\text{Rigid Tables} + \text{Variable Tables}}$$

A1.2.4 Feature: Functional Modifier NWDSEN

Function: NWDSEN acting on a table name produces the number of words per table entry.

Deletion: It is hard to conceive of any use for NWDSEN!

Resolution: Retain if U GE .25 where

$$U = \frac{\text{Number Occurrences of NWDSEN}}{\text{Tables}}$$

A1.2.5 Feature: String

Function: Provides a means of declaring packed two dimensional arrays.

Retention: Control of packing in (two-dimensional) arrays is not possible in J3 in any direct way.

Deletion: By use of BIT and BYTE it is possible to indirectly achieve the effect of String.

Note: This is further evidence for arguing that the concept of recurrent structures should exist separately from the concept of controlled storage allocation.

Resolution: Retain if U GE .10 where

$$U = \frac{\text{Strings}}{\text{Tables with Defined Packing}}$$

A2. Function Module: Referencing Data (Nucleus)

Function: To provide the facilities for specifying data as arguments for data operators.

Retention: The facilities in this module provide for referencing parts of data aggregates and parts of items. (Simple items --- those not declared as parts of aggregates --- are referenced simply by supplying their identifier names).

Modification: Subscripting is seen as required since (at least) arrays are to be retained. Referencing components of items is not seen as required, as described below.

Resolution: Retain referencing capability.

A2.1 Feature: Functional Modifier BIT

Function: To reference a substring of an item. (The substring is regarded as an unsigned integer.)

Deletion: This feature can be performed by overlaying the item to be modified with an integer item (if it is not itself integer), assigning the overlaying item to a temporary integer item, and multiplying/dividing the temporary item by powers of 2 to simulate shifting.

Retention: If a great deal of substring manipulation is required, the method shown above is clumsy, and BIT can be a significant convenience to the programmer.

Note: BIT is particularly useful when manipulating data that does not fall into any of the J3 item type classes (numeric, literal, boolean, status) but rather represent a bit string which carries some especially encoded information. "Equipment Codes" used to command peripheral equipment falls into this class.

Resolution: Retain if U GE .03, where

$$U = \frac{\text{Occurrences of BIT}}{\text{Numeric + Literal + Boolean + Status}}$$

A2.2 Feature: Functional Modifier BYTE

Function: To reference a substring of a character item. (The substring itself is treated as a character item.)

Note: The properties ascribed to BIT apply to BYTE in much the same way.

Resolution: Retain if U GE .03 where

$$U = \frac{\text{Occurrences of BYTE}}{\text{Literal}}$$

A2.3 Feature: Functional Modifiers CHAR/MANT

Function: CHAR references the characteristics of a floating point item; MANT references the mantissa of a floating point item. (CHAR is treated as a signed integer; MANT as a fixed point item with no integer bits).

Note: It is possible in principle to use BIT to access either the characteristic or mantissa of a floating point item. In fact, BIT is not defined (in the standard) with respect to a floating point item. This is because floating point word formats vary from machine to machine and in computers with no floating point hardware must be simulated - characteristic and mantissa might even be stored in different words.

Deletion: It is possible to obtain the characteristic and mantissa by assigning to a fixed point item and using, if necessary, successive divisions by two.

Retention: CHAR/MANT are much more convenient if this function need be performed fairly often.

Resolution: Retain both if U GE .03 where

$$U = \frac{\text{Occurrences CHAR + Occurrences MANT}}{\text{Floating Point}}$$

A2.4 Feature: Subscripting (Nucleus)

Function: To reference an elementary item in an array, table, or string.

Retention: Required, since almost all operations in J3 with respect to single items.

Resolution: Retain.

Note: Use of functional modifier POS to point to records in a file is a kind of subscripting. However, POS is taken up separately under input/output.

A2.5 Feature: Functional Modifier ENTRY

Function: To reference a particular entry of a table.

Deletion: This function may be performed by referencing each item in the entry of a table individually.

Retention: In manipulating tables it is often convenient to pick up the entire entry at one time. Example: Sorting the entries of a table with say ten items/entries involves iterative interchanging of entries. Without ENTRY, the programmer would have to write ten exchange statements to do the job; with ENTRY, only one.

Resolution: Retain if U GE .02 where

$$U = \frac{\text{Occurrences ENTRY + Occurrences ENT}}{\text{Tables}}$$

A2.6 Feature: ENT or ENTRY Notation

Function: ENT may be used instead of ENTRY.

Delete: It is ridiculous to have two so similar ways of calling out the ENTRY functional modifier.

Resolution: Retain that primitive which is used more often in the majority of applications; delete the other.

(Note: If the ENTRY function is deleted, this, of course, has no meaning.)

A3. Function Module: Representing Data in the Standard J3 Memory

Function: To provide a model of a computer memory and mechanisms for specifying value representation and storage allocation with respect to that model. (The memory model (JAQ, pp. 42, 43) is required to provide a frame of reference for definitions of BIT, BYTE, defined packing, etc.)

Deletion: The classic argument is that high-level languages should be free of memory representation considerations and storage allocation mechanisms for two reasons. First, the programmer should be relieved of such considerations; second, such considerations are always machine dependent, hindering the rapid transfer of a program from one system to another.

Retention: High-level languages frequently fail to do the job because programmers are not provided with facilities for effectively making use of memory and other computer-specific resources. (It is interesting that JOVIAL has much more comprehensive facilities of this kind than other high level languages.)

Resolution: This function module will not be deleted if one or more of its submodules is retained. Deletion of certain submodules can conceivably effect the depth of detail of the memory model, however, and certain far ranging modifications may ensue. It is important with this function module to determine how well the J3 memory simulates the actual machine memories on which JOVIAL is implemented --- low usage rates of features in this module might imply inability to simulate the memory model as opposed to lack of requirement for the facilities. This topic will be brought up at interviews.

A3.1 Feature: Octal Constant

Function: Provides the ability to specify data values solely in terms of bit patterns.

Retention: Such a function provides a user convenience in programming efforts where special encoding schemes may be required.

Deletion: Setting up bit patterns may be accomplished by clever use of integer values based on appropriate powers of 2.

Modifications: Octal is a number base particularly suited to computers with word and word substructure bit lengths evenly divisible by $3(2^3 = 8)$. Many third generation computers --- notably System 360 and Spectra 70 --- have word and word substructure bit lengths evenly divisible by 4, making a hexadecimal number base more convenient for specifying bit patterns. Hexadecimal, or a more general way of declaring bit patterns, may be desirable.

Resolution: Retain functional capability if majority of applications have one or more usages of octal constants. Implementation of modification will be recommended on

the basis of interview responses.

A3.2 Feature: Representation of Negative Integers in Signed Magnitude Form

Function: The Standard requires that negative numbers be represented in signed magnitude form, i.e., the magnitude of a signed integer, fixed point, or floating point number is represented in terms of the same bit pattern whether or not the number is positive or negative.

Deletion: Many computers represent magnitudes of negative numbers in either 1's complement or 2's complement form. With such computers, a great deal of support code must be generated to simulate the signed magnitude representation.

Retention: Using signed magnitude representation makes for consistency when overlaying or using the BIT functional modifier.

Example:

8 bit word negative 3 in signed magnitude form is 1 000 0011. Using BIT($ 1,7$) (word) produces 000 0011 which has the same representation as +3 (excluding sign bit). In 1's complement form, -3 is 1 111 1100 and the result of the same BIT expression produces 111 1100, not 3.

Resolution: Depending on responses to the JAQ and opinions expressed at interviews, signed magnitude may be deleted as a Standard requirement.

A3.3 Feature: Basic (Table) Structure Attribute

Function: Permits a given item for all entries of a given table to be placed in consecutive words ("parallel") or all items of the same entry to be allocated to consecutive words ("serial").

Retention: Sometimes parallel structuring is a convenience since it is possible to pass all items of a kind as an array to a procedure which does not need to be aware of the particular structure of the table.

Deletion: Parallel/Serial imposes a burden on the compiler in the sense that it must generate code reflecting two different allocation schemes.

Resolution:

(1) Let U1 = $\dfrac{\text{Parallel Tables}}{\text{Parallel Tables + Serial Tables}}$

(2) Let U2 = 1.0 - U1

If either U1 or U2 GE .10, retain parallel/serial. Otherwise retain parallel or serial only depending on which has the greatest usage rate.

A3.4 Feature: Ordinary Packing

Function: Permits the specification of table packing --- "Medium" or "Dense" in degree.

Retention: Tables are allocated 1 word per item per entry with "No" packing. (More than 1 word in the case of long literals.) If sizes of items are such that they can be packed more than 1 to a word, great economies of space may be obtained. For example, if two words/entry can be saved by packing, a 100 entry table can cause a savings of 200 words.

Deletion: Packing generally requires generation of more code to reference an item than if that item were non-packed. Ordinarily, savings in storage space more than make up for the extra code produced.

Modification: It is unclear that the"byte sharing" requirements stated for ordinary packed tables (JAQ p.56) can be meaningful for all computers. It might be desirable to drop these requirements.

Resolution: (Dropping requirements described under modification will be recommended if responses to JAQ indicates that few versions comply.) Retain if U GE .05 where

$$U = \frac{\text{Medium Packed + Dense Packed}}{\text{Tables}}$$

A3.5 Feature: Defined (Table) Packing

Function: To allow the programmer to specify precisely (relative) word number in the entry and starting bit in that word for a given item.

Note: In general, the same arguments regarding ordinary packing apply here also.

Retention: The programmer may create his own allocation scheme, taking advantage of machine features possibly overlooked by the compilers ordinary packing allocation scheme.

Deletion: Defined packing is obviously very machine dependent, while ordinary packing is not.

Resolution: Retain if U GE .10 where

$$U = \frac{\text{Defined Tables}}{\text{Tables}}$$

A3.6 Feature: OVERLAY

Function: Permits the programmer to direct sequence of allocation to memory for J3 items and aggregates, and common relative origins. (Absolute origins are discussed in A3.7)

Retention:

(1) OVERLAY permits space saving because it permits using the same area of memory for different sets of data at different times in the execution of a program.

(2) More often, OVERLAY is used to interpret the same data in different ways during the course of execution, or to give explicit names to substructures of aggregates.

Example (A)

```
ITEM XX I 48 S $
ITEM YY F $
OVERLAY AA = BB $
```

enables data in the same container to be interpreted as integer (using XX) and floating point (using YY) at different points in the program's execution.

Example (B)

```
ARRAY MATRIX  2  2  I  $
ARRAY COLUMN'1  2  F  $
ARRAY COLUMN'2  2  F  $
OVERLAY MATRIX = COLUMN'1,
                 COLUMN'2  $
```

permits assigning a name to each column of MATRIX.

(3) Note that OVERLAY is unique in J3 in performing the functions ascribed to it.

Modification: In keeping with the "uniqueness of purpose" principle, it might be better to have separate statements to handle the space saving and reinterpretation/substructure naming aspects of OVERLAY.

Resolution: If the majority of applications use OVERLAY at least once, it should be retained.

A3.7 Feature: Absolute Origin

Function: JOVIAL provides the ability to establish absolute machine addresses for data (in OVERLAY statements) and, at the implementor's option, permits for the specification of absolute origins for programs.

Deletion: Specification of absolute addresses is inflexible because it prohibits relocation of data or program blocks. Relocation is highly desirable since it allows the programmer to "assemble" program and data blocks in a wide variety of ways. Further, it would seem that absolute origins are highly sensitive to operating system environment. A time shared system might not be able to support absolute origins since data and program segments are dynamically relocated.

Resolution: Retain or delete according to insights gained from interview discussions.

A3.8 Feature: Functional Modifier 'LOC.

Function: 'LOC acting as a statement label or data identifier produces the absolute machine address of the statement label or identifier.

Note: The same comments made about absolute origins apply to 'LOC.

Resolution: Retain or delete based on insights gained at interviews.

A4. Function Module: Numeric Items and Arithmetic Operations (Nucleus)

Function: To provide for representation of numeric data and arithmetic operations on the numeric data.

Retention:

(1) The model of numeric data in J3 corresponds closely to the kinds of numeric data programmers must deal with in parameterizations of command and control problems. In particular, J3 permits representation of numbers which
- contain no fractional part (type integer)
- contain a fractional part, the number of fractional digits of which are known throughout the course of program execution (type fixed point)
- contain a fractional part, the number of digits of which may vary during program execution (type floating point).

(2) Numeric operations automatically manipulate the fractional point, paralleling point manipulation in computations performed independently of computers, and freeing the programmer of embedding scaling in his code.

(3) The notational form for coding numeric formulas is (almost) identical to common algebraic notation and as such provides an ease of expression for the command and control programming community.

Modification:

(1) In J3, type integer is a special case of type fixed point (scale = 0). Arithmetic with integer items is treated in the same way as arithmetic with fixed point items. Thus, division of one integer by another may result in an integer and fractional part. This property is desirable, but differs from

division as performed by most computers; machine division results in a separate quotient and remainder. To provide greater clarity of J3 arithmetic processing for programmers (many of whom think of integer arithmetic as performed on the machine level) it may be desirable to eliminate the integer item type as a data entity and to use fixed point, scale zero, exclusively.

(2) The implied number base in J3 is 2. It is conceivable that justification exists for introducing base 10 into the computational facilities as well. (Numbers are often input and output in base 10; also some computers have hardware base 10 arithmetic capability.)

Resolution: Retain

(1) Integer, Fixed Pointing, Floating Point numeric items
(2) Arithmetic operators +, infix -, *, /, ** (with automatic scaling property)
(3) Algebraic notational scheme (not including prefix + or -)

(Implement modification on the basis of interview discussions)

Note: +, -, *, / are unique operators in J3. ** is not unique --- it may be represented in terms of more elementary operators --- but including ** in the nucleus set of operators provides significant user convenience.

A4.1 Feature: Numeric Item Size Attribute (Nucleus)

Function: Provides necessary information for the scaling of numeric items in computations.

Retention: Required.

Modification:

(1) J3 numeric specifications are given in terms of the size attribute. If the item is signed, the number of magnitude bits is "Size - 1"; otherwise number of magnitude bits = size. Number of magnitude bits means precision. It would appear more to the point to specify precision, not size, when declaring items. (The sign attribute must be specified in any event.)
(2) Precision in J3 cannot exceed the number of magnitude bits in one (J3 memory) word. Frequently greater precision than that available in one word is required (this is true of floating point as well as fixed point computations) and the object computer has double precision hardware arithmetic. Greater utility from J3 might be realized if facilities were incorporated to provide for precision greater than that available in one word, in such instances.

Resolution: Retain functional capability. (Implement modifications on the basis of discussions at interviews.)

A4.2 Feature: Fixed Point Item Scale Attribute

Function: To provide the number of fractional bits of a fixed point item for use in scaling.

Retention: Required.

Resolution: Retain.

A4.3 Feature: Range Attribute

Function: To control the scaling of intermediate calculations for the purpose of maintaining maximum precision.

Retention: The automatic scaling facility in J3 must establish the scales of intermediate results in terms of precisions of the items used in obtaining the intermediate results. Sometimes scaling may cause a dropping of precision needlessly.

> Example: Two unsigned fixed point items, AA and BB have integer parts of 5 and 3 bits respectively. The automatic scaling facility would allocate an intermediate integer part of 8 bits for the product AA * BB since the resultant integer produced by multiplying the maximum numbers registerable in 3 and 5 bits requires 8 bits for its representation. However, if the maximum integer part values of AA and BB throughout execution are 17 and 6, the maximum value ever taken by the product (102) requires only 7 bits. Therefore 1 more bit may be picked up for use in registering the fractional part of the product -- greater precision is achieved. (The sum integer and fractional digits = the number of magnitude digits in the computer word.)

Deletion: The same effect can be achieved by breaking up involved computations into a series of smaller ones and judiciously assigning "intermediate" results to appropriately scaled "temporary" numeric items.

Resolution: Retain if U GE .10 where

$$U = \frac{\text{Number of Occurrences of RANGE}}{\text{Integer + Fixed Point}}$$

A4.4 Feature: Sign Attribute

Function: Permits the user to specify whether or not sign information is to be carried with an integer or fixed point number.

Note: To delete the sign attribute would mean to delete "unsigned" as an attribute value (i.e., all numbers automatically signed). To delete "signed" as an attribute value would prevent representation of negative numbers --- such a situation is clearly intolerable.

Retention:

(1) The ability to declare numbers "unsigned" enables the programmer to save a binary digit when packing tables.
(2) With respect to computers which do not represent negative numbers in signed magnitude form, declaring a number as unsigned can effect savings in the amount of code generated to support simulation of signed magnitude.

Deletion: All numbers are treated as signed by computers when calculations are being performed.

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Number Occurrences "U"}}{\text{Fixed Point + Integer}}$$

A4.5 Feature: Prefix + and -

Function: Prefix + has only a notational effect. Prefix - enables the programmer to change the sign of a numeric quantity which precedes.

Note: Prefix + and - are referred to here in the context of numeric formulas, not as used in front of numeric constants in presetting. In presetting, prefix - must be available.

Deletion:

(1) The effect of prefix - can be obtained by multiplying the numeric quantity by -1, i.e., -AA is the same ans MINUS'ONE * AA, where the variable MINUS'ONE has been preset to -1.
(2) Prefix + has no effect whatever.

Retention:

(1) It is clearly more convenient to use prefix -, as seen from the example above. Moreover, prefix - is commonly accepted in algebraic notation.
(2) Prefix + is notationally convenient to the programmer when he is trying to emphasize signs for himself and for other readers of his code.

Modification: Prefix - takes precedence in J3 over exponentiation. In algebraic notation, exponentiation takes precedence. In order to make JOVIAL formula notation comply more closely with algebraic notation it is advisable to consider altering the precedence of prefix -.

Resolution: Retain Prefix + and - if U GE .01 where

$$U = \frac{\text{Number of Occurrences of Prefix - and +}}{\text{Numeric Formulas}}$$

(Implement modification on the basis of interview responses.)

A4.6 Feature: Exponentiation Notation

Function: Provides for representing the exponentiation operator as ** or as (* *).

Deletion: Two ways of representing exponentiation are not required; one way should be deleted. Note that ** has been widely used in high level programming language and is probably familiar to the majority of programmers. The brackets (* and *) are notationally pleasing in that they can be thought of as superscript marks (exponents are often represented as superscripts) and have a parallel in JOVIAL to subscript marks (S S).

Resolution: Retain whichever is used most frequently in the majority of applications; delete the other.

A4.7 Feature: Absolute Value

Function: To provide a built-in operator in the language to take absolute values of all numeric item types.

Deletion: It is possible to accomplish the same effect in terms of other (nucleus) JOVIAL features.

Retention: A formal mechanism for taking absolute values in J3 is a programmer convenience.

Resolution: Retain if U GE .01 where

$$U = \frac{\#\text{ Occurrences of ABS or (/ /)}}{\text{Numeric Items}}$$

A4.8 Feature: Absolute Value Notation

Function: Provides for specifying the absolute value operator either as the prefix operator ABS or in terms of the brackets (/ /).

Deletion: Two different notations for absolute value operations are not required.

Resolution: Whichever notation is used more often in the majority of applications will be retained. The other will be dropped. (Meaningful only if absolute value built-in mechanism is retained.)

A4.9 Feature: REM and REMQUO

Function: REM and REMQUO permit the programmer to compute, respectively, remainder

and the remainder and quotient produced by the division of two integers. (Remainder and quotient appear as integers.)

Note: As remarked above, the division of two integers by the / operator result in a fixed point number.

Retention: The functions performed amount to modulo operations. If such a function needs to be performed often, it is a user convenience to maintain REM and REMQUO in the standard.

Deletion: The functions can be performed easily in terms of other (nucleus features). In fact, if the programmer requires a great deal of usage of this feature, he can easily write his own PROC'S REM and REMQUO (see JAQ p. 125 and p. 122).

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurrence of REM and REMQUO}}{\text{Integer Items}}$$

A5. Function Module: Dual Items and Dual Arithmetic Operations

Function:

(1) To provide for representing aggregates consisting of two integer or fixed point components. Components are not named separately; reference by name may be made only to the aggregate as a whole.
(2) To provide automatic mechanisms for performing arithmetic (and assignment) operations on each component separately without specifying separate operations for each component.

Retention: It would appear that dual items and arithmetic have been designed with two dimensional vector or complex number representation and manipulation in mind. This function module would be very convenient to programmers involved in simple vector manipulation, since component by component operations are performed automatically.

Deletion:

(1) This function can be performed without the formal dual mechanisms provided, by simply declaring and operating on each component separately.
(2) Parameterization of vectors using dual items is restrictive since only fixed and integer items are permitted and these must not exceed in size half the number of bits in the computer word.
(3) It is not possible to conveniently reference an individual component of a dual item, since individual components do not have names. Presumably, BIT would have to be used to reference a component when it is desired to manipulate one component only.

(4) The Dual item is restrictive in the sense that only two component vectors may be represented.

Modification:

(1) A generalization of the dual item/arithmetic function is apparent and should perhaps be considered at interviews. It may be desirable to have the ability to write arithmetic expressions in which operands are whole arrays or tables (unsubscripted). Operators in such arithmetic expressions would be cause component by component operations to take place automatically.

Example: In such a system, the programmer would write:

```
ARRAY  AA  10  F  $
ARRAY  BB  10  F  $
ARRAY  CC  10  F  $
AA = BB * CC  $
```

to produce the effect of

```
FOR I = D, 1, 10  $
AA ($ I $) = BB ($ I $) * CC ($ I $)  $
```

(Note: PL/1 has such a facility as this.)

(2) Still other generalizations, differing from the one described above, are possible. "Structure" arithmetics other than the component by component variety are frequently required in command and control programming. Matrix algebra is such a structured arithmetic, and might be introduced into the language in terms of a data type MATRIX.

Example: The programmer might write:

```
MATRIX AA  3 3 F $
MATRIX BB  3 3 F $
MATRIX CC  3 3 F $
AA  =  BB * CC  $
```

The operator * would have the effect of producing the matrix product of BB and CC. (Type VECTOR with operators CROSS and DOT might similarly be introduced.)

Resolution: Retain if U = .10 where

$$U = \frac{\text{Dual}}{\text{Numeric} + \text{Literal} + \text{Status} + \text{Boolean}}$$

(Recommendations to implement modifications will be based on interview responses. Low usage rate of dual items/arithmetic and high interest in structure arithmetics may be interpreted as meaning that dual arithmetic mechanisms specified in the Standard are too restrictive to cope with user problems.)

A6. Function Module: Testing Operations (Nucleus)

Function: Provides the facilities for relational and other testing.

Retention: Fundamental

Resolution: Retain

Note: Testing in J3 produces a Boolean variable and is therefore connected to the logical manipulation subsystem described in A7. The logical subsystem has not been declared as nucleus and may be deleted. This would have no substantial effect on Testing Operations, save the context in which testing operators might appear (only in IF, IFEITH/ORIF statements, not in Boolean Formulas.)

A6.1 Feature: Relational Operators and Simple Relational Expressions (Nucleus)

Function: Provides for comparing item data by use of relational formulas. Relational operators are EQ, NQ, LS, GR, LQ, GQ. When evaluated, relational formulas produce "true" if the operands in fact stand in the indicated relations; otherwise the value produced is "false."

Retention: Required. (Note: Only three of the six relational operators provided are actually required (EQ, LS, GR), but it is felt that inclusion of the other three provides a worthwhile convenience.)

Modification: If the operands are Hollerith, only EQ and NQ have meaning in the standard. This is equivalent to saying that there is no collating sequence available in J3, thus making less-than/greater-than alphanumeric comparisons impossible. It would appear desirable to introduce a Hollerith collating sequence into the language so that meaning can be given to LS, GR, LQ, GQ.

Resolution: Retain

(Implementation of the modification will be recommended depending on user response at interviews).

A6.2 Feature: "Chained" Relational Feature

Function: Provides the capability for writing relational formulas in chains, e.g.,
3.0 LS X ' COORDINATE LS 4.0

Retention: Chained relational expressions constitute a notational convenience in that their use saves writing and parallels commonly used mathematical notation (e.g., $3 < \chi < 4$).

Deletion: The effect produced by this feature can be obtained by using several simple relational formulas (e.g., 3.0 LS X COORDINATE and X 'COORDINATE LS 4.0)

Resolution: Retain if U GE .03 where

$$U = \frac{\text{Chained Relational formulas}}{\text{Relational Formulas}}$$

A6.3 Feature: Functional Modifier ODD

Function: Tests the rightmost binary digit of a numeric item or loop variable and produces the result "true" if the digit is one.

Retention: ODD is useful in determining whether or not a numeric or loop variable is odd or even.

Deletion: The function can also be easily performed by other facilities in the language.

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurrence of ODD}}{\begin{array}{l}(\text{Integer} + \text{Fixed Point} + \text{Floating Point} + \\ 1 - \text{factor FOR} + 2 - \text{factor FOR} \\ \quad + 3 \text{ factor FOR})\end{array}}$$

A7. Function Module: Boolean Items and the Logical Operators AND, OR, NOT

Function: Enables both the representation of logical (Boolean) quantities (which take either the value "true" or the value "false") and use of the common logical operators.

Retention: The primary purpose of incorporating a Boolean Algebra subsystem in J3 is to aid in decision making. "Decision making" as used here means altering the flow of program control, as with an IF statement, depending on the value currently taken by one or more variables. In J3, the operands of IF statements are Boolean variables or formulas. The inclusion of a complete Boolean algebra subsystem (Boolean items; AND/OR/NOT) helps the programmer specify complex decision making conditional expressions. Boolean algebra is a coherent system for performing this job, is known to many programmers, and can be easily learned by others.

Deletion: The complete Boolean subsystem may be a significantly more powerful a tool than is required to cope with the great majority of decision making problems. Implementation of the complete Boolean Subsystem, in that case, places an added burden on J3 compilers.

Resolution: This function module will be deleted if both features in it (Boolean Items and AND/OR/NOT) are deleted.

A7.1 Feature: AND, OR and NOT

Function: Perform the common logical operations.

Retention: AND, OR, and NOT provide convenience to programmers in writing complex IF statements, e.g., IF AA EQ BB AND CC LA 3.0 $ is easier to write -- and because it is all on one "line" -- may be easier to read -- than the two separated IF statements

```
IF AA EQ BB $
IF CC LS 3.0 $
```

Deletion: The function can be performed by repetition of IF statements containing only simple (or chained) relational formulas.

Resolution: Retain if U GE .02 where

$$U = \frac{\text{Occurrences of AND, OR, NOT}}{\text{Relational Formulas}}$$

A7.2 Feature: Boolean Items

Function: "Closes" the Boolean subsystem by permitting the formal representation of logical quantities.

Retention: The results of evaluations of complex relational formulas can be saved for later interrogation by assigning the relational formula value to a Boolean Item:

```
AA = BB LS CC AND (DD EQ 3 OR XX GR 7) $
```

The Boolean item itself can be operated on by AND, OR, and NOT or assigned the logical constant values "true" (1) or "False" (0).

Deletion: Programmers generally think of the Boolean item device as a "flag". Clearly other item types can be made to play the flag role in circumstances such as the one described above:

```
IF BB LS CC AND (DD EQ 3 OR XX GR 7) $
    AA = 1 $
```

Resolution: Retain if U GE .02 where

$$U = \frac{\text{Occurrences of Boolean Items}}{\text{Numeric + Literal + Status}}$$

A8. Function Module: Assignment (Nucleus)

Function: To assign values to variables

Retention: Fundamental

Resolution: Retain this functional capability

A8.1 Feature: Presetting

Function: Enables assignment of a value to a named item, table, array, or string at compilation times.

Deletion: The presetting function can be performed by execution time assignment statements.

Retention: Presetting saves the programmer writing and does away with assignment statement code used (only once!) at execution time.

Resolution: Retain all presetting capabilities (item, table, array, string) if

U GE .02 where

$$U = \frac{\text{Occurrences of presetting (all types)}}{\text{(Item Decs + Array Decs + Table Decs + String Decs)}}$$

A8.2 Feature: Rounded Numeric Assignment

Function: When a numeric item is declared with a round specifier "R", any numeric quantity assigned to it which has greater

Retention: If one or more variables in a program is such that rounding should always precede assignment, this is a useful feature.

Deletion: Rounding can be performed by the programmer by adding an appropriate power of 2 to the numeric quantity to be assigned to the item, and by performing the (usual) truncated assignment.

Modification: Rounding is more conveniently thought of as an attribute associated with the assignment or arithmetic operators. That is, it is important to have the facility to round when assigning to a variable on some occasions and not on others.

Resolution: Retain functional capability if U GE .05

$$U = \frac{\text{Occurrences of Round Specification}}{\text{Integer + Fixed + Floating Point}}$$

(Implementations recommended on the basis of responses at interview)

A8.3 Feature: Exchange Statement

Function: Enables the programmer to cause values of two items (or table entries) to be exchanged.

Deletion: Exchanging can be performed by three assignment statements, i.e.,

```
AA = BB $
is equivalent to
TEMP = AA $
AA = BB $
BB = TEMP $
```

Retention: A formal exchange statement is convenient since it saves writing. It also saves declaring the Item TEMP. Particular advantages are enjoyed when the exchange operands are ENTRY's of a table.

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurrences of Exchange Statements}}{\text{Assignment Statements}}$$

A9. Function Module: Program Structure and Execution of Program Structures (Nucleus)

Function:

(1) Provides for the formation of simple statements into aggregates. These aggregates are compound statements, closed subprograms and programs.
(2) Provides mechanisms for directing the sequence of execution of program structures.

Retention: This function module is required.

Modification: The execution of program statements and structures in J3 takes place in a basically serial mode. This reflects the fact that computers, if not all processes simulated on them, have up until recently behaved in a serial manner, performing one operation at a time. (It is important to note that many basically serial computers are in fact capable of carrying on input/output in parallel with other operations.) Currently computers are being designed and built which are capable of performing many operations in parallel. It is perhaps desirable to include formal mechanisms in J3 to direct parallel execution. (Such facilities have been specified for PL/1).

Resolution: Retain this functional capability. (Implementation of the modification will be recommended on the basis of response at interviews.)

A9.1 Feature: Compound Statements (Nucleus)

Function: Provides the facility for delimiting the scopes of subprocesses.

Retention: The formation of compound statements is not the only way imaginable for delimiting scopes. However, it is felt that it is a good way of performing this function

and is fundamental to the block-like structure of J3.

Resolution: Retain.

A9.2 Feature: GOTO (Statement' label) (Nucleus)

Function: Enables the programmer to unconditionally transfer control to another part of the program structure, other than the next statement in sequence.

Retention: Fundamental.

Resolution: Retain.

(Note: RETURN and TEST are unconditional transfer features also but are discussed in the context of PROC's and FOR-loops).

A9.3 Feature: STOP and STOP Statement' label

Function:

(1) If STOP $ appears in an independently compiled subroutine, it causes control to pass to the calling program.
(2) If it appears in an independent program, control is passed to the monitor, if there is one.
(3) If there is no monitor, the computer may idle, and if restarted, resume at the next statement'label.

Modification: It is clear that some facility for halting the sequence of executions, either physically, or by passing control to a monitor, or both, ought to be included in the light of user requirements. The following notions need to be considered:

(1) There should be one and only one way of returning control from a closed subroutine. Such a facility is provided by the RETURN statement. It would be desirable to have STOP in an independently compiled closed subroutine return control to the monitor, not to the calling program. (Sometimes unrecoverable errors are detected in subordinate routines, and the only sensible course of action is to abort the job in the subordinate routine.)

(2) Halting the computer is highly environment sensitive. In real time systems halting may be paramount to disaster.

Resolution: The disposition of STOP must await JAQ responses and discussions at interviews.

A9.4 Function Module: Conditional Transfer of Control (Nucleus)

Function: Permits the sequence of control to be altered conditionally, based on the status of one or more variables.

Retention: Required.

Modification: The facilities of this module --- IF, IFEITH/ORIF, Item Switch, and Index Switch --- apply to testing the current values of programmer declared data. When executed, some branching always takes place on the basis of these values. Other kinds of branching facilities are sometimes desirable. Such facilities cause branching based on the current status of the hardware --- interrupts, underflow, overflow, etc. Often when testing for machine conditions, no branching takes place immediately when the testing instruction is executed. Rather, a monitoring mechanism is activated, and any branching will occur at some later time when the monitoring mechanism senses the specified condition. (The monitoring mechanism works in parallel with other processing.) It may prove useful to incorporate formal capabilities for conditional branching of this kind in J3.

Resolution: Retain functional capability. (Recommendation to implement the modification will be made depending on response at interviews.)

A9.4.1 Feature: IF (Nucleus)

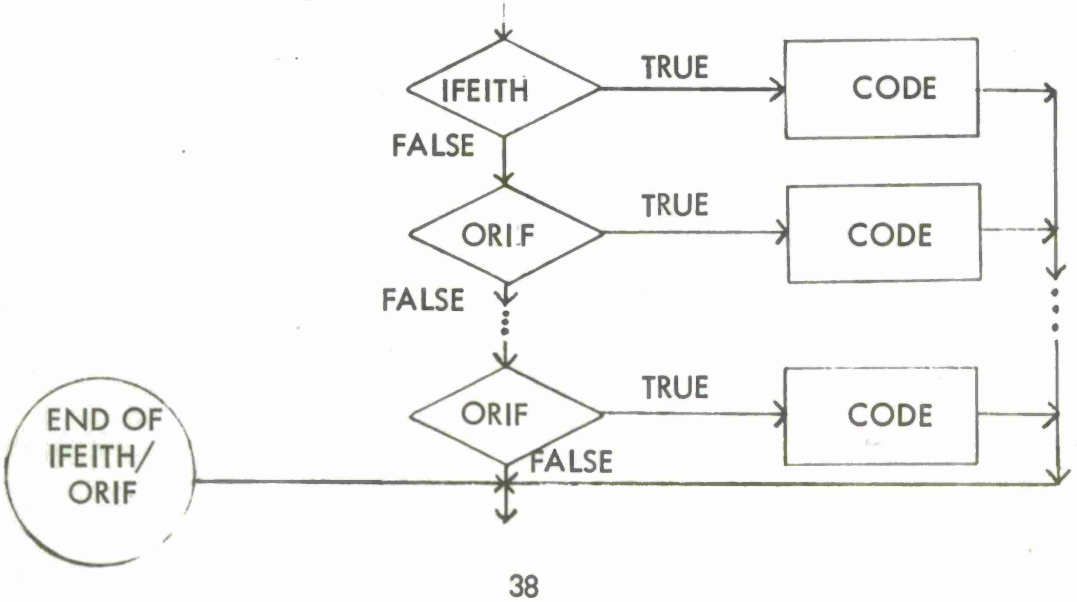
Function: Functionally and notationally the simplest feature provided for effecting conditional transfer of control.

Retention: Fundamental.

Resolution: Retain.

A9.4.2 Feature: IFEITH/ORIF

Function: Provides the facilities for creating a chain of IF-like statements (IFEITH or ORIF). At execution each IF-like statement is operated in sequence until a "true" condition obtains or until the end of the chain is reached. When a true condition obtains, the associated code is operated and control is automatically passed to the end of the IFEITH/ORIF chain. The diagram illustrates:



Retention: This feature is particularly convenient when a decision making situation like that shown in the diagram is encountered because the branching code at the end of the "true" code is automatically provided. (The programmer does not have to insert GOTO end ' of' chain statements.)

Deletion: This function can be performed with IF statements.

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurrences of IFEITH}}{\text{Occurrences of IF}}$$

A9.4.3 Feature: Index Switches

Function: Control is passed to the kth statement label of a list of statement labels if the subscript associated with the switch name has the value k when the switch is invoked. (Index switches are invoked by GOTO switch ($ subscript $).)

Deletion: This function can be performed by IF statements. The IF statements would be used to test for values of a flag (the subscript). When a match occurred, control would be passed to the statement associated with the matched flag value.

Retention: Clearly the index switch is more convenient, since use of it saves the writing of all the IF statements. (Also, many computers have a hardware indexed jump capability. Index switch invocations compiled into indexed jump instructions make for very fast execution time.)

Resolution: Retain if U GE 0.05 where

$$U = \frac{\text{Occurrences Index Switches}}{\text{IF Statements}}$$

A9.4.4 Feature: Item Switches

Function: Provides an automatic "table lock-up and jump" facility in that the value of a given variable is matched against each value of a list and when a match is made, control is passed to a statement named by an associated statement label.

Deletion: This function can be performed in terms of IF statements (testing the given value against each of a set of values) and GOTO statements (which pass control to the associated statement). (If FOR loops and Index switches are available, the function can be performed by them with much less code than the Index switch approach.

Retention: Clearly, item switches are more convenient to the programmer notationally. (Also, item switch code may be compiled into "search" instructions on machines which

have such hardware capabilities, producing very fast object code.)

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurrences Item Switches}}{\text{IF Statements}}$$

A9.5 Function Module: FOR Statements

Function: Provides for activating loop variables, and the iterative alteration of the sequence of execution.

Retention: As described for each feature below.

Deletion: As described for each feature below.

Resolution: This function module will be deleted if each feature comprising it is deleted; otherwise it will be retained.

A9.5.1 Feature: 3-Factor (Complete) FOR Statement

Function: Provides for declaring assigning an initial value to, incrementing and testing a loop variable for the purpose of iteratively executing a simple or compound statement.

Deletion: This function can be performed by use of declared integer variables, assignment statements, simple arithmetic expressions and IF statements (i.e., the programmer may make his own loop in terms of J3 statements.)

Retention: Iteration is fundamental to computer programming. The Complete FOR loop saves the programmer setting up his own loop variable, incrementation and testing code. Also, the notational form of the FOR statement is similar to that used by mathematicians when they specify iterations. (On computers with hardware index registers, it is possible to have the compiler "translate loop-variables to index registers" so that this hardware feature may be put to good use.)

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurences of 3-factor FOR}}{\text{Arrays + Tables}}$$

A9.5.2 Feature: 2-Factor FOR Statement

Function: Provides for declaring, assigning an initial value to, and incrementing a loop variable.

Deletion: As with Complete FOR Statements, the 2-factor Statement can be implemented in terms of other J3 statements.

Retention: 2-factor FOR Statements have advantages similar to the 3-factor FOR. They also permit the programmer to do his own limit testing which can be a convenience when such testing is out of the ordinary.

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurence 2-factor FOR Statements}}{\text{Tables + Arrays}}$$

A9.5.3 Feature: 1-Factor FOR Statement

Function: Provides a means of activating a loop-variable and assigning a value to it. The loop variable may then be used as a subscript in the next statement or in a formula used to evaluate a parameter for a subsequent FOR Statement.

Retention: It can be convenient to use a loop variable declared in a 1-factor FOR Statement instead of declaring an integer variable to perform the job.

Deletion: The function can be performed by declaraing an unsigned integer variable and giving it a value either by presetting it or by means of an assignment statement.

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurences 1-factor FOR Statement}}{\text{Tables + Arrays}}$$

A9.5.4 Feature: TEST

Function: Permits transfer of control from any point within a 2- or 3-factor loop statement to the code which controls the iteration.

Retention: Since the iteration control code is supplied automatically, it cannot be named by the programmer and, therefore it cannot be referenced in a GOTO statement.

Resolution: This feature must be retained if 2- or 3- factor FOR statements are retained.

A9.6 Function Module: Closed Subprograms and their Execution (Nucleus)

Function: Enables the programmer to declare "self-contained" subprograms and to operate them remotely (i.e., control is passed to the closed subprogram from a calling program, and when the closed subprogram terminates execution it returns control to the calling program at the point immediately following the envoking statement).

Retention: Although it is possible to write J3 programs without this capability, not to have it places a great burden on the programmer. Closed subprograms allow for the modularization of programs. Modularization is a fundamental programming tool, especially in the implementation of large scale systems.

Resolution: Retain this functional capability.

A9.6.1 Feature: Procedure (Nucleus)

Function: Provides the ability to declare subprograms with a variety of formal input and output parameters and to invoke such subprograms with actual parameters automatically substituted for the formal ones. (Also provides function declaration and invocation capability.)

Retention: It is felt that of the two kinds of closed subprograms available, this is the more powerful, and should be retained.

Modification: It is sometimes convenient to have alternate entrances into procedures. Alternate entrances have different names and possible different parameter lists. Although it is possible to effect alternate entrances by means of disjunctive procedures, the alternate entrance approach permits the sharing by each entrance of blocks of code and (internally declared) data common to all.

Resolution: Retain Procedures (and Functions). (Recommendation to implement modification will be made on the basis of responses at interviews.)

A9.6.2 Feature: CLOSE

Function: Provides the ability to declare a subprogram which, like a Procedure, may be invoked remotely. Unlike a Procedure, no linkage of parameters may take place, and all data declared internally to the CLOSE may be referenced by external program segments.

Deletion: It is possible to perform the function of CLOSE by using a Procedure with no calling sequence parameters, except as noted below. (External program segments may reference internally declared data if that data is declared externally of the Procedure.)

Retention: It is possible to declare a CLOSE within a FOR-loop, thus making loop-variables available to the CLOSE directly. (In such cases, the CLOSE may not be invoked from outside the loop.) If a Procedure were used, the loop-variables would not be directly accessible within the body of the procedure, though it is possible to pass loop-variable values into the Procedure, by first assigning to an integer variable.

Resolution: CLOSE will be deleted, if substantial agreement can be reached at interviews and if no cause for retaining it can be shown.

A9.6.3 Feature: RETURN

Function: Permits exiting a Close or Proc from a point preceding the physical end of the subprogram.

Retention: RETURN passes control to the "exit" code associated with the closed subprogram. This code cannot be named with a statement label for reference by a GOTO. (It is possible to place a labelled statement at the physical end of the closed subprogram which does nothing of importance and to GOTO that statement to return. However, such a dummy statement produces code and requires execution time.)

Resolution: Retain.

A9.7 Feature: Program Declarations (Nucleus)

Function:

(1) Provides START/TERM delimiters for use by the compiler.
(2) Provides the ability to declare that a symbol being used in a program is the name of an independently compiled closed subprogram ('PROGRAM declaration.)
(3) Provides the ability to compile a CLOSE as an independent subroutine. (CLOSE ident $ START Statements TERM).

Retention:

(1) Delimiters such as START/TERM are required to inform the compiler of the scope of a compilation.
(2) The facility to provide for closed routines to be independently compiled and linked together at "load" time is extremely important in large scale command and control programming systems. Without such a facility modularization becomes extremely difficult.
    - the ability to declare that a program is to be compiled as an independent subroutine, and
    - the facility to inform the compiler that a symbol being used is the name of an externally compiled subroutine

    are felt to be required.

Modifications:

(1) It is desirable to compile Procedures as independent subroutines, as well as Closes. (If Closes are deleted, this capability would, of course, have to be implemented.)
(2) Declaration of a symbol as a program name should be made in the COMPOOL, not in the program being compiled. ('PROGRAM declaration should be deleted.)

Resolution: The ability to compile closed subprograms independently and to declare their

names as names of independently compiled programs should be retained. (Implementation of modification will be recommended based on interview responses.)

A9.8 Feature: Direct Code

Function: Permits the programmer to insert segments of machine (Assembly Language) instruction into a J3 program.

Retention: This facility is valuable to the programmer when he cannot perform a required data processing function by use of J3. It is the hope of this study that a Standard will be recommended which will provide enough capability for the programmer that he would never require use of Direct Code, but it is also recognized that some tasks are so machine dependent as to be non-supportable in a higher level language. (I/O drivers are an example). It is possible to argue that an operating system be capable of linking routines written entirely in J3. The Direct Code feature has certain advantages over this technique, however, in that it allows direct cross-referencing of J3 declared data with the machine code (via the ASSIGN Statement).

Resolution: Retain.

A10. Function Module: Input/Output (Nucleus)

Function: Provides mechanisms for performing input and output operations.

Retention: Input/Output, in some computer systems, is performed by means of (machine language) subroutines. It is felt, however, that I/O processing is such a fundamental operation that formal mechanisms should be available to perform this function. In J3, I/O is centered around the concept of files -- this concept has gained wide acceptance in the computing community and should be retained. However, it is felt that certain flaws exist in the I/O model described in AFM 100-24. The next paragraphs (modifications) describe these flaws and will serve as a nucleus for discussions at interviews.

Modifications:

(1) It is convenient to maintain the notion of files as a logical entity -- a collection of records -- separate from the notion of physical devices on which the file information is stored. J3 mixes these two notions together.
(2) The identifier which names file as a set of records also serves as a status variable indicating state of the I/O device to which the file is attached.
(3) File declarations require device dependent information to be specified once and for all. It is sometimes convenient to assign different devices to the same "logical" file at different points in the course of execution.
(4) It is required that a maximum number of records in the file be stated when the file is declared. For files written on discs and drums this is not an unreasonable requirement. For tape files, it might be impossible to even know this information.
(5) Files have to be declared Hollerith or Boolean to indicate whether code conversions (Hollerith) need to take place. (Some systems have different internal and

external octal representations of characters). In some systems, this facility involves the kind of parity checking system used also, though it is possible that code conversions and parity may be two separate attributes in other systems.

(6) The use of functional modifier POS as currently defined creates certain difficulties.
   - If "POS = O" is used to "rewind" the files, what meaning does "POS = O" have when attached device is a tape with two separate files (separated by an end-of-file mark) and it is desired to position at the 0th record of the second file?
   - For random access files, how can POS be "linked" with track/sector directories, established by the programmer? That is, is it possible for the programmer to define his own accessing algorithm for the purpose of organizing the space available in the storage medium?

(7) Files in J3 must be either in the INPUT or OUTPUT mode when they are being manipulated. With random access files it is often desired to read a record, modify its contents, and write the record back in the same place from where it was fetched. It would appear desirable to have a third mode, "INOUT", for permitting such operations.

(8) File manipulation in J3 is at the "device level". That is, only physical records in the file are ever manipulated; further, read and write operations are only initiated in J3 and must be monitored by the programmer "to completion." In some applications it is desirable to work at the "logical file" level, where it is possible to have "logical" records automatically blocked into and deblocked from physical records and where multiple buffering can be automatically arranged. On the "logical file" level, INPUT and OUTPUT refer to getting and putting logical records (from buffers) and are automatically "monitored through to completion."

(9) No means are provided in J3 I/O for converting character string representations of data to and from internal binary representations. Such a capability is very important for inputting punched card data and outputting BCD data to the printer.

Resolution: Retain basic functional capability and file concept (recommendations for implementation of modifications will be made on the basis of interview responses and JAQ responses.)

A11. Function Module: Naming Data and Program Structure (Nucleus)

Function: Provides the mechanisms for naming data and program structures.

Retention: Required.

Modifications:

(1) J3 requires that names be unique, except that names in Procedures may also be used in "outer" programs with no ambiguity. Some programming languages

permit repetition of names with ambiguities resolved by qualification (as in COBOL and PL/1).

(2) When data are declared in J3, attributes are given values which are unalterable throughout the course of execution. It is at times desirable to alter attribute values associated with a name at different points in the program.

Resolution: Retain.

(Recommendations for implementation of modifications may be made on the basis of interview responses.)

A11.1 Feature: Item Declaration, Form 2

Function: Allows the programmer to specify an (initial) constant instead of attribute values as in Form 1 of the item declaration. The attribute values are determined from the constant.

Retention: Provides a convenience to the programmer in that it saves him formally writing attribute values.

Deletion: Form 2 item declarations are not required and the same function can be performed by Form 1.

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurrences Form 2}}{\text{Occurrences Form 1}}$$

A11.2 Feature: Multiple Statement Labels

Function: Allows the programmer to give more than one name to a statement.

Retention: It is sometimes convenient to be able to refer to a statement by more than one name. (This feature is to statement names as OVERLAY is to data.)

Deletion: Obviously, only one label is required.

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurrences Multiply Label Statements}}{\text{Total Procedure Statements}}$$

A12. Function Module: Definitional Facilities

Function: Provides programmers with the facilities to make definitions (and "abbreviations") for often used language constructs.

Retention: Definitional facilities provided a user convenience in that they save repetitively writing code.

Deletion: None of the features in this function module is required; all functions performed by them may be performed by the programmer if he "fills in details" himself, rather than allowing the compiler to do it.

Modification: A current trend in high-level programming languages is the development of powerful definitional systems which parallel the concept of macros available in many assembly languages. Macros in high-level languages allow for the introduction of new data structures and operators in terms of those already existing in the (nucleus) language and they also permit the addition of new syntactic forms into the grammar of the language. Macro facilities have two distinct advantages:

(1) They permit language extensions in directions selected by applications Programmers, not by language designers trying to satisfy all needs, now and in the future, with one omnibus language.
(2) Extensions are made in terms of the nucleus language, thus doing away with the need for developing whole new compilers each time a language is to be updated.

It may be desirable to introduce macro capabilities into J3.

Resolution: This functional module will be deleted only if all features comprising it are deleted. (Recommendation to implement modifications will be made on the basis of interview responses.)

A12.1 Feature: The DEFINE Directive

Function: Enables the programmer to give a single name to a string of J3 symbols. All subsequent references to that name in the programmers code are then automatically replaced by the compiler with the symbol string.

Retention: This feature saves the user repetitive writing of symbol strings.

Deletion: Clearly, the programmer can code the symbol strings himself.

Modification: A possible useful extension to the DEFINE directive would be the addition of an "argument substitution facility" as exists in Procedures. Example:

```
DEFINE SUM (A1, A2, A3)
"A1 = A2 + A3 $ " $
```

when "called" by SUM (ZZ, XX, YY) $ would result in the J3 statement

```
ZZ = XX + YY $
```

Resolution: Retain this feature if U GE .005 where

$$U = \frac{\text{Occurrences of DEFINE}}{\text{Total Number of Statements}}$$

(Recommendation to implement modification will be made based on user response at interviews. Note that a high degree of interest in the modification may over-ride deletion of DEFINE, since this interest would imply that DEFINE, as currently specified, is not adequate to address the problem, but that the problem is indeed worth addressing.)

A12.2 Feature: The MODE Directive

Function: Allows the programmer to specify an automatic set of data attribute values for identifiers which are not formally declared.

Retention: This facility saves the programmer from formally declaring a set of identifiers, all of which have the same attributes. Also, the MODE directive aids the compilation process in the sense that no identifier is left undefined.

Deletion: This function can be performed by the programmer making his own declarations.

Modification: The MODE directive is restrictive in that only one item type can be automatically declared. A possible extension would be to provide the capability for more than one automatic declaration based on, say, the first letter of the identifier (as in FORTRAN).

Resolution: Retain if U GE .02 where

$$U = \frac{\text{Number of Items Declared by MODE}}{\text{Number of Items}}$$

(Recommendation to implement modification on the basis of responses at interview.)

A12.3 Feature: Defining Ordinary Packing Specifiers

Function: Allows the programmer to define to the compiler the meanings of M (medium packing) and N (dense packing) in terms of defined table packing specifications.

Deletion: The meaning of this feature as described in AFM 100-24 is very imprecise.

Resolution: The disposition of this feature will have to be determined at interviews.

A12.4 Feature: Like Table Declaration

Function: Allows the programmer to declare a table the entry items of which are identical to those of another table, without formally declaring the entry items.

Retention: This feature can be very useful where it is desirable to have a "working temporary" table identical in structure to a table being manipulated.

Deletion: The programmer may make his own working table declaration.

Resolution: Retain if U GE 0.05 where

$$U = \frac{\text{Like Table Declarations}}{\text{Tables Declared with Formal Declarations}}$$

A12.5 Feature: Functional Modifier ALL

Function: Permits the abbreviation of initial, increment, and final values in the use of a FOR loop to process all entries of a table:

```
FOR  I = ALL (TAX'TABLE)  $  means
FOR  I = NENT (TAX'TABLE) -1, -1, -1  $
```

Retention: This feature saves the repetitive writing of NENT ( ) -1, -1, -1.

Deletion: Not required.

Resolution: Retain if U GE .05 where

$$U = \frac{\text{Occurrences of ALL}}{\text{3-factor FOR Statements}}$$

A12.6 Feature: COMPOOL

Function: Provides the ability to maintain a "dictionary" of declarations which are referenced automatically by the compiler.

Retention: In large command and control systems, a COMPOOL-like tool is extremely useful. Data referenced extensively in one or more programs of the system need be declared only once, saving repetitive writing of declarations in each program. Also, if data specifications are changed, they need be changed only once.

Modification: It is felt that the J3 Standard should specify more precisely what kind of declarations may appear in the COMPOOL.

Resolution: Unless good reasons are presented to the contrary, a precise specification of what declarations may appear in the COMPOOL will be included in the final report, based on interview responses.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Data Dynamics, Incorporated 9800 S. Sepulveda Boulevard Los Angeles, California 90045	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

APPROACH FOR CHANGE JOVIAL EVALUATION PROJECT

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Final Report

5. AUTHOR(S) *(First name, middle initial, last name)*

William M. O'Brien

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1968	53	2

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
F19628-68-C-0301 b. PROJECT NO.	ESD-TR-68-455
c. d.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Command Systems Division, Electronic Systems Division, Air Force Systems Command, USAF, L G Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

The Approach for Change document was produced to establish the criteria for evaluating features of the JOVIAL (J3) computer programming language as specified in AFM 100-24. The document contains arguments for retaining, deleting, or modifying JOVIAL features including a methodology for resolving the arguments.

DD FORM 1 NOV 65 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Programming Language Evaluation JOVIAL Evaluation						