AN ABSTRACT OF THE THESIS OF

JONATHAN AXFORD COLBY for the MASTER OF SCIENCE
in COMPUTER SCIENCE presented on APRIL 1, 1976

Title: The FLEX/REFLEX Paradigm and Its Application to Computer Program Systems

Abstract approved: Redacted for Privacy

Paul Cull

The FLEX/REFLEX paradigm is applied to the description of a computer program system. The paradigm is shown to be relevant and appropriate to computer program systems and to advantageously display and structure the general hierarchical characteristics of computer program systems. Program systems characterized in the paradigm are described both holistically and mechanistically at each hierarchical level. This, together with an explicit description of and distinction between input quantities and output quantities, permits effective and intuitively logical management of even large systems. The problems of program synthesis, modification, testing and validation, debugging, and documentation may be decomposed into "sub-problems" associated with each module in the hierarchical structure. Only problems relevant to the holistic behavior of the module need to be considered. Problems associated with the behavior of the elements of the module are dealt with at the next lower hierarchical level, when the element itself is considered as a module. The paradigm is also shown to be consistent and compatible with top-down structured programming.
The FLEX/REFLEX Paradigm and Its Application to Computer Program Systems

by

Jonathan Axford Colby

A THESIS submitted to

Oregon State University

in partial fulfillment of the requirements for the degree of

Master of Science

June 1977
APPROVED:

Redacted for Privacy

Associate Professor of Computer Science
in charge of Major

Redacted for Privacy

Chairman of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis is presented  April 1, 1976

Typed by Eva Hofenbredl for Jonathan Axford Colby
# Table of Contents

## Introduction

1. Development of the Flex/Reflex Paradigm for Ecosystems
   - Historical Background
   - Ecosystem Characteristics

2. The Hierarchical Approach
   - The Nature of Hierarchies
   - Hierarchies of Systems
   - Simulation Investigation of Hierarchical Systems

3. Elaboration of the Flex/Reflex Paradigm

4. Application of the Flex/Reflex Paradigm to Program Systems
   - Other Approaches to Program Systems
   - Definition of Program Systems by the U-C Structure
   - Advantages of the U-C Structure

5. Application to a Particular Program System
   - The Input Procession of the Flex Processors
   - Conventions and Notation
   - Resolution Level

6. Results and Discussion
   - Relation to Top-Down Structured Programming

7. Bibliography
TABLE OF CONTENTS (Continued)

APPENDICES

<p>| Appendix A: FLEX/REFLEX Input Processor Description | 38 |
| Appendix B: Description of FORTRAN Variables       | 73 |
| Appendix C: Keyword Parameter Directory Array      | 77 |
| Appendix D: Input Language Definition              | 80 |
| Appendix E: Error Messages and Codes               | 85 |
| Appendix F: Edit Characters and Rules              | 87 |
| Appendix G: System Diagrams                         | 88 |
| Appendix H: Program Listing                         | 91 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>la.</td>
<td>The system activity, the external quantities at the instant of time ( t ) and all past and future instants.</td>
<td>12</td>
</tr>
<tr>
<td>lb.</td>
<td>The principal quantities of time ( t ) are identified as those visible when the mask is imposed on the system activity at time ( t ).</td>
<td>13</td>
</tr>
<tr>
<td>2.</td>
<td>Schematic representation of the FLEX algorithm.</td>
<td>14</td>
</tr>
<tr>
<td>3.</td>
<td>The FLEX and REFLEX system forms.</td>
<td>16</td>
</tr>
<tr>
<td>4.</td>
<td>Schematic representation of hierarchically stacked modules.</td>
<td>17</td>
</tr>
<tr>
<td>5.</td>
<td>The mechanistic model form as a set of coupled subsystems.</td>
<td>18</td>
</tr>
<tr>
<td>6.</td>
<td>Overview of the FLEX/REFLEX simulation package.</td>
<td>24</td>
</tr>
<tr>
<td>7.</td>
<td>System diagram of the FLEX (1, 2, or 3) processor.</td>
<td>25</td>
</tr>
<tr>
<td>8.</td>
<td>Algorithmic diagram of the FLEX/REFLEX input processor.</td>
<td>89</td>
</tr>
<tr>
<td>9.</td>
<td>Conceptual diagram of the FLEX/REFLEX input processor.</td>
<td>90</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>S,1,1,0 Couplings - Inputs</td>
<td>46</td>
</tr>
<tr>
<td>2.</td>
<td>S,1,1,0 Couplings - Outputs</td>
<td>47</td>
</tr>
<tr>
<td>3.</td>
<td>S,1,1,1,0 Couplings</td>
<td>50</td>
</tr>
<tr>
<td>4.</td>
<td>S,1,1,1,1,0 Couplings</td>
<td>54</td>
</tr>
<tr>
<td>5.</td>
<td>S,1,1,1,1,1,0 Couplings</td>
<td>56</td>
</tr>
<tr>
<td>6.</td>
<td>S,1,1,7,0 Couplings</td>
<td>66</td>
</tr>
<tr>
<td>7.</td>
<td>S,1,2,0 Couplings</td>
<td>69</td>
</tr>
</tbody>
</table>
A particular definition of a general system is necessarily motivated by the problem, or class of problems being investigated. From a given definition of a system one may proceed to develop a particular general systems theory that will permit the solution of a large class of problems. However, there are certain to be problems unapproachable by that theory which would be solvable by use of another general system theory arising from a different definition of a system. The intimate relationship between the definitions of the system and the problems to be solved requires that the traits and properties of the problem, and the perspective from which they are viewed by the investigator be examined prior to the selection or development of the general system theory that will be used to investigate the problems. The interrelationship of problem, definition, and theory follows from the nature of the modeling process. Modeling is organizing, structuring, and ordering knowledge. The value judgements, or criteria, by which some information is excluded as irrelevant, and varying degrees of importance are assigned to the information selected, directly determine the model structures that result. The value judgements, however, are necessarily made with respect to the particular problem objectives. What you find depends on how you go looking.
The FLEX/REFLEX paradigm provides for model structures and techniques of system analysis that are both appropriate and effective when applied to certain of the problems of computer program systems: verification, validation, definition and management of program structures. This paper, after elaborating the paradigm, will discuss and demonstrate the application of the paradigm to computer program systems in general and as applied to a specific example.
DEVELOPMENT OF THE FLEX/REFLEX PARADIGM FOR ECOSYSTEMS

Historical Background

The FLEX/REFLEX paradigm\(^1\) was developed under the direction of Dr. Scott Overton in response to the problems posed by ecological systems investigation undertaken by the Coniferous Forest Biome of the International Biological Program\(^2\). A basic understanding of coniferous forest ecosystems was sought through a broad interdisciplinary approach. A systems analysis and modeling approach was used to identify and define the major components and processes and their interrelationships and characteristics (Franklin 1972). The FLEX/REFLEX paradigm was developed to encompass the perceived system characteristics of ecosystems and systems in general, and to provide a framework for the construction of well defined and logically consistent models which explicate how a system works according to the current theory.

Ecosystem Characteristics

The investigation team viewed the coniferous forest ecosystem as entities consisting of coupled self organizing and resilient assemblages (Overton 1972) in which each assemblage, or system, is viewed as an object in the context of and coupled with its environment, as well as

\(^1\)Paradigm in the sense of Kuhn (1970): "a universally recognized achievement that for a time provides model problems and solutions to a community".

\(^2\)Supported by National Science Foundation Grant Number GB 3680X to the Coniferous Forest Biome, Ecosystem Analysis Studies, U.S./International Biological Program.
being viewed as a coupled collection of subsystems. Compared with the classical scientific approach to systems as being closed and studied in isolation, ecosystems are viewed as being open systems, investigated in the context of their environment. Emphasis was placed on the holistic properties of each identified system and its hierarchical relationship to both the subsystems which compose it and the system in which it is a component. This perspective resulted in two distinct approaches to a given system. The holistic approach is concerned with the exhibited external behavior of a system. The mechanistic approach is concerned with the coupled relations of explicitly identified subsystems. Underlying this dual approach is the assumption that all the properties of a system are not recognizable as direct properties of its parts but are derived instead from the coupled interactions of the parts.

As a simple example of this premise, consider a system of two tuning forks, one with a frequency of 540 cps and the other with a frequency of 544 cps. The whole system produces three tones; one at 540 cps, one at 544 cps, and one at 4 cps - the beat frequency. The system may also be viewed as two subsystems, one producing a 540 cps tone and the other a 544 cps tone. It is quite valid to investigate each subsystem independently of the other, but it is obviously invalid to conclude that the whole system behavior is a simple sum of the independent behavior of its subsystems.

Requiring each system to be described holistically as well as mechanistically provides for logical consistency and completeness. The behavioral characteristics that are strictly functions of a particular subsystem are completely accounted for in the mechanistic description
of the subsystem. The behavioral characteristics that are functions of the interactions of the behaviors of the subsystems are accounted for by the holistic description of the system. A different partitioning of processes (that is, identification of different subsystems) would result in different mechanistic descriptions for the subsystems (since they would exhibit behaviors different from the previous partitioning) but the holistic description would remain the same.
THE HIERARCHICAL APPROACH

The Nature of Hierarchies

Hierarchies, and orderings in general, are necessarily made with respect to particular criteria. Viewing the hierarchy as a tree network of nodes and branches, each node has associated with it a set of criteria that define, in a mutually exclusive manner, the branches leading from it. Selecting and ordering the criteria creates the hierarchy. The hierarchy may then be imposed upon a particular body of information in an attempt to elicit and display previously hidden characteristics or traits associated with a particular node or sub-segment of the hierarchy. As an example, dichotomous keys are commonly used for plant identification by biologists. A particular individual plant is "fitted" to a hierarchy of physical characteristics, usually with the goal of determining the genus and species of the individual.

Obviously, the processes of creation and application of hierarchies are not disjoint, for a judicious selection of criteria requires careful consideration of the nature of the traits sought and the nature of the criteria available. The judicious selection of criteria is the process of modeling. Models are hierarchies "fleshed out" with a particular body of information.

Hierarchies may be generally characterized as static or dynamic, depending on whether the classification criteria are physical or behavioral, respectively. The above example of the key is a static hierarchy. Direct association and statistical methods are usually the most appro-
appropriate for the investigation of static models. Determination of the environmental parameters of various species of diatoms by statistically correlating the physical factors with the species is an example. The static hierarchy would be the one level (node) classification of physical characteristics of the environment. In static models, emphasis is on physical traits and their classification according to a criteria.

Dynamic models emphasize the interrelationships of the physical traits, or processes, and the processes are classified according to the criteria. Because of the time varying nature of most processes, mathematical analysis (the methods of linear algebra, calculus, etc.) and simulation techniques are usually the most appropriate methods for investigating dynamic models. As an example, consider the following model of the behavior of an automobile. For simplicity, consider only the existance of motion, speed and acceleration and not the direction. The motion, depends primarily upon two sub-processes, the transfer of mechanical energy and the conversion of chemical energy to mechanical energy, or internal combustion. The latter in turn depends essentially upon the carburation of fuel and its movement through the combustion chamber, and the combustion of the fuel. Assuming the auto to be in working order, competently operated and its engine running, the essential controlling factors for the transfer of mechanical energy are the brakes and the clutch. Together they determine whether motion exists - the auto is stopped or moving. The speed and acceleration are functions of the transfer of chemical to mechanical energy. The acceleration depends upon the change of the rate at which fuel is moved through the
combustion chamber, while the speed of the auto for a given rate of fuel movement, depends upon the efficiency of the combustion.

Several things should be noted about the fore-going example. First, the hierarchical structure is applicable not only to automobiles but generally to the motion characteristics of the class of land vehicles powered by internal combustion engines. The hierarchical structure may be appropriately applied to automobiles and diesel trucks and locomotives. Second, the model is very simple, for the classification and ordering dictated by the hierarchical structure is quite coarse. The auto is considered only while in operation, and relative to the effects of processes identified. Processes such as acceleration due to gradual engagement of the clutch and brakes are considered to be insignificant. A more detailed and complex model might consider these processes, and would generally yield more detailed and precise answers about local behavior, although, as the model becomes more complex and detailed, understanding of the overall behavior may be diminished. The model yields almost no information about the nature of the vehicle. It is required a priori to be a vehicle powered by an internal combustion engine, but the investigation of the model yields information only about the nature of the motion of the vehicle. A model based on a static hierarchy of physical characteristics would be appropriate for investigating the nature of the vehicle to determine whether it is a locomotive, automobile, truck, etc. and if an automobile, what make, and perhaps even who owns it.

For a static model, if direct association of traits of the particular model with those of the hierarchy does not directly yield the
desired information, because of variance or inexactness of measurement, the application of statistical methods would be appropriate. The dynamic model requires an analytic technique, such as the solution of a set of differential equations, or a stimulation technique, such as a computer simulation model.

Since any node in the hierarchical tree may be characterized by more than one criterion, and since each level or echelon of nodes is characterized by its own criteria, it is quite possible to have a hierarchy that is neither particularly static or dynamic. There is nothing wrong with this per se but as noted previously, the problems that may be appropriately investigated are dependent upon the criteria of the hierarchy. If the purpose of it is to investigate behavior, the more dynamic the hierarchy, the better the chances for success.

Hierarchies of Systems

As the terms subsystem and supersystem imply, systems are commonly structured hierarchically. Most general systems theories assume or require by definition that systems may be structured hierarchically. Also basic to the nature of systems, and recognized also by most general systems theories, are a system's behavioral properties. Systems are required to exhibit some form of behavior at least under some conditions. Consequently, by definition, a hierarchy of systems must be dynamic.

Since the definition of systems is generally recursive, and the decision as to what is the environment is arbitrary, a hierarchy of systems may itself be perceived as a system. We thus speak of hierarchical
systems. And here we often get into trouble. It is quite common to identify a static hierarchy of components of a system and refer to the system as being hierarchical. The components most always exhibit behavior, but since the primary basis for selecting the component was a physical trait, the system is not structured hierarchically. The components have been organized hierarchically, but this static organization is not germane to the investigation of the system's behavior.

The critical question to ask about a system is "what happens?", not "what's there?". What's there is only important as a result of asking the question "what influences or causes what happens?" In the automobile example, no mention was made of an engine in the descriptions of the processes. The engine does not make the automobile go. What happens in the engine is what makes the automobile go. In ecosystems, it is not the species found that determines the behavior of an ecosystem, but rather what the species do that determines ecosystem behavior.

Simulation Investigation of Hierarchical Systems

The effectiveness with which simulation techniques may be applied to systems is highly dependent on the way a system is structured. The curse of dimensionality prohibits much investigation of the response surface of even a very small model of a system. For example, a system of nine parameters, each with only three values, would require $3^9 = 19,683$ simulations over the selected time interval. However, if the system has been structured in a dynamic hierarchy, in which components of the behavior being investigated have been isolated onto say two subsystems of five parameters each, a substantial reduction in simulations is
achieved. Only $2 \times 3^5 = 486$ simulations would be needed. This may still be beyond practical limits and one would have to settle for an incomplete investigation. However, far more has been gained than otherwise.
ELABORATION OF THE FLEX/REFLEX PARADIGM

The following is a brief description of the FLEX/REFLEX paradigm as developed by Overton et al. (1973), Overton (1975) and White and Overton (1974). The general systems theory of George Klir (1969) was found to be generally consistent with the identified characteristics of ecosystems and the needs of investigating ecosystems through modelling. Klir identified five alternate definitions of controlled systems. The three that are relevant, and form the basis for the FLEX/REFLEX paradigm, are:

1. Definition by the set of external quantities and their resolution. The controlled system is a given set of external quantities and a given set of output quantities, both at a given space-time resolution.

2. Definition by permanent behavior. The controlled system is a given time invariant relation that holds between the principal quantities associated with the latest values of output quantities on the one hand and the set of all principal quantities on the other.

3. Definition by the real Universe-Coupling structure. The controlled system is a given set of controlled elements, together with a set of couplings directed from output quantities of one element to input quantities of other elements, including the environment.

The first two definitions, considered as one and referred to as the definition by permanent behavior, lead to the holistic model form. The
third definition is referred to as the definition by universe-coupling (UC) structure and leads to the mechanistic model form.

The concept of principal quantities is essential in the definitions and are defined in the following fashion. Klir first defines the external quantities as the system outputs $Y$ and the system inputs $Z$. A mask is then chosen which blocks most but not all of the past, current, and future values of the external quantities for a selected time instant $t$. The principal quantities are the external quantities taken at the time instants which remain visible through the mask. The behavior is defined as the time invariant relation between the instantaneous output quantities and the rest of the principal quantities. This is illustrated in Figures 1a and 1b.

<table>
<thead>
<tr>
<th>External Quantities</th>
<th>....</th>
<th>t-3</th>
<th>t-2</th>
<th>t-1</th>
<th>t</th>
<th>t+1</th>
<th>t+2</th>
<th>....</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z_1$</td>
<td>....</td>
<td>$Z_1_{t-3}$</td>
<td>$Z_1_{t-2}$</td>
<td>$Z_1_{t-1}$</td>
<td>$Z_1_t$</td>
<td>$Z_1_{t+1}$</td>
<td>$Z_1_{t+2}$</td>
<td>....</td>
</tr>
<tr>
<td>$Z_2$</td>
<td>....</td>
<td>$Z_2_{t-3}$</td>
<td>$Z_2_{t-2}$</td>
<td>$Z_2_{t-1}$</td>
<td>$Z_2_t$</td>
<td>$Z_2_{t+1}$</td>
<td>$Z_2_{t+2}$</td>
<td>....</td>
</tr>
<tr>
<td>$Y_1$</td>
<td>....</td>
<td>$Y_1_{t-3}$</td>
<td>$Y_1_{t-2}$</td>
<td>$Y_1_{t-1}$</td>
<td>$Y_1_t$</td>
<td>$Y_1_{t+1}$</td>
<td>$Y_1_{t+2}$</td>
<td>....</td>
</tr>
<tr>
<td>$Y_2$</td>
<td>....</td>
<td>$Y_2_{t-3}$</td>
<td>$Y_2_{t-2}$</td>
<td>$Y_2_{t-1}$</td>
<td>$Y_2_t$</td>
<td>$Y_2_{t+1}$</td>
<td>$Y_2_{t+2}$</td>
<td>....</td>
</tr>
<tr>
<td>$Y_3$</td>
<td>....</td>
<td>$Y_3_{t-3}$</td>
<td>$Y_3_{t-2}$</td>
<td>$Y_3_{t-1}$</td>
<td>$Y_3_t$</td>
<td>$Y_3_{t+1}$</td>
<td>$Y_3_{t+2}$</td>
<td>....</td>
</tr>
</tbody>
</table>

Figure 1-a. The system activity, the external quantities at the instant of time $t$ and all past and future instants. (Overton 1975)
Figure 1-b. The principal quantities of time t are identified as those visible when the mask B imposed on the system activity at time t. After Klir (1969) in Overton (1975).

The FLEX/REFLEX paradigm defines state variables (X) for convenience in the calculation of the instantaneous values of the system outputs and in many cases these will be identical. Occasionally, internal variables may be useful for conceptual or computational reasons. Memory variables (M) are defined as selected past values of state variables and input variables. Output variables (Y) are then identified as functions of the state variables (X), input variables (Z), and memory variables (M) (See Figure 2a). It should be noted that the customary state variable definition includes both the memory variables and state variables of the FLEX/REFLEX paradigm.

Figures 2b and 2c are expansions of the functional generator and memory of Figure 2a. Two parameter vectors R and B, as well as Z, X, and M are available for use by the functional generator. Whether the time invariant relation referred to in the system definition is implemented as a system of difference equations or differential equations depends only on the conceptualization of time t as either discrete time.
Figure 2. Schematic representation of the FLEX algorithm of the general paradigm of sequential systems as modified and explicated from Klir (1969) in Overton (1975).
with the solution exact for the chosen time resolution, or as continuous
time with the solution being given by an appropriate differential tech-
nique.

The directed couplings between two systems $S_i$ and $S_j$ are designated
as $C_{ij}$ and $C_{ji}$ where $C_{ij}$ is the set of output variables of $S_i$ which are
the inputs of $S_j$, $C_{ij} = Y_i \rightarrow Z_j$, and similarly $C_{ji} = Y_j \rightarrow Z_i$. The
coupling with the environment of a coupled collection of systems is
accounted for by a ghost system that integrates the inputs and the out-
puts of the proper subsystems. The ghost system contains all the
features identified in Figure 2 except the $f$ functions. These are
replaced by the subsystems. The ghost system is described by the
REFLEX convention, and the proper subsystems are described according to
either the REFLEX convention or the FLEX convention depending on whether
or not they themselves are ghost systems (See Figure 3).

Since the universe coupling structure permits a finer resolution
model, the temporal resolutions of the proper subsystems must be integral
fractions of the temporal resolution of the whole system. Each subsystem
has an integral resolution parameter $q_i$ that specifies the number of
time steps of the subsystem for each step of the whole system, $S_0$. $S_0$
"manages" the subsystems. $S_0$ is updated by its subsystems at their
temporal resolutions, which allows it to manage the couplings of the
subsystems by providing for the regulation of flow relations by both
donor and receiver elements. Consequently, the need for rigid sequential
processing of subsystems is eliminated. Furthermore, since a model of
either the FLEX or REFLEX forms involves exactly the same specifications
Figure 3. The FLEX and REFLEX system forms. (Overton 1975).
of inputs, outputs and resolution, either form may be used interchangeably, achieving a hierarchical modularity (See Figure 4).

Figure 4. Schematic representation of hierarchically stacked modules. The two systems are equivalent in terms of total system behavior but emphasize different aspects at finer resolution. REFLEX (Ghost) modules are represented by (0).

The model structure of the FLEX/REFLEX paradigm is also expressed as the FLEX/REFLEX algorithm. It is one way of characterizing the paradigm in terms of a set of sequential mathematical operations (White et. al. 1974). The algorithmic structure emphasizes the ghost system's regulation of subsystem coupling. This leads to two equivalent forms of diagramming, identified as the conceptual and algorithmic form (See Figure 5). The algorithm establishes the basis for the FLEX/REFLEX computer processors, FLEX1, FLEX2, and FLEX3, which are general purpose computer program systems that handle the computer simulation of FLEX/REFLEX models.
Figure 5. The mechanistic form is viewed as a set of coupled subsystems. Emphasis is on the particular couplings between the subsystems (a), or on the nature of coupling regulation (b) (White and Overton 1974).

The FLEX/REFLEX paradigm was developed to cope with the problems of investigating ecosystems. To date, several dozen ecosystem models of varying complexity and resolution have been successfully developed (for examples, see McIntire et. al. 1975 and Overton and White 1974). Unpublished work by the author and White demonstrated the successful application of the paradigm to the problems of regional (state wide) land use planning and resource allocation.

Ecosystems comprise a very large class of systems, and any paradigm applicable to them would necessarily be considered a general system paradigm even if it applied to no other systems. However, those traits and characteristics of ecosystems considered in the development of the FLEX/REFLEX paradigm are not unique to ecosystems, but are also common to a much larger class of systems. This was deliberate. While the practical considerations of ecosystem investigation required that the
paradigm be developed and applied to ecosystems, much effort was given to fashion the paradigm so that it might suitably be applied to a much broader class of systems.
APPLICATION OF THE FLEX/REFLEX PARADIGM TO PROGRAM SYSTEMS

During the development of the FLEX/REFLEX processors, which was roughly concurrent with the development of the FLEX/REFLEX algorithm, many insights into the general nature of systems were spawned by characteristics of the computer programming system in which the processors were developed. This is not surprising. Computer programming systems are very comfortably defined by the universe coupling structure and by their permanent behaviors. The application of the FLEX/REFLEX paradigm follows directly.

Other Approaches to Program Systems

The FLEX/REFLEX approach to program systems is by no means the only approach, nor is it the only fruitful approach. Several theories of varying degrees of generality have been successfully applied to program systems. Among them are the abstract theories of finite and infinite automata, in particular the theory of turing machines (Minsky 1967) and the theory of Formal Languages (Hopcroft and Ullman 1969). Each of these theories provides different insights into the nature of program systems. The insights (with the resultant understanding) visible from the structure prescribed by one theory are often not apparent under the structure of the other theory. The theory of Turing Machines, among other things, addresses itself to the problem of solveability - what a program system could and could not solve. The theory of Formal Languages, among other things, addresses itself to the nature of the structure of the language of a program system. There are many other
general system theories that have been proposed and elaborated, such as those by Wymore (1967) and Mesarovic (1969). An attempt to apply these theories to programs systems might prove to be highly successful.

**Definition of Program Systems by the U-C Structure**

We shall define a computer program system to be the machine instructions electronically encoded and resident in the computer and its associated peripheral hardware. The computer is considered to include the hardware and any software implementation of machine instructions (such as a monitor or supervisor implementation of input-output instructions). Also, if the computer is timeshared, only the time slices allotted to the particular program system will be considered. This view is consistent with the usual perspective from which programs are written in a particular language. From the above definitions, we see that a program language listing, or program source, is a model of the program system, with an assembler or compiler and a loader providing the mapping from the program source to the program system.

Definition of the program system by the universe coupling structure is very consistent with the conventional view of program systems. Elements of the U-C structure are the individual machine instructions, each of which is defined holistically and mechanistically; holistically according to its permanent behavior (the particular operation, such as addition, transfer, etc.) and mechanistically according to the hardware microprograms that implement the operation. The coupling variables are comprised of the various registers and memory locations of the computer.
Because of the obvious advantages of simplicity, notational conventions and our general familiarity with it, the program source will be investigated as the model of the program system and the terms program system and program source may be used interchangeably.

Advantages of the U-C Structure

The advantages of economy and manageability afforded by hierarchical structures have been recognized from the outset of computer program development. Hierarchical structures developed rapidly, starting with macro instructions and moving through the development of languages such as ALGOL, FORTRAN, and COBOL, each of which has the capabilities (SUBROUTINE-CALL, PERFORM, etc.) to generate system elements. The assembly of programs into data processing systems (business payroll systems, statistical analysis systems, etc.), the development of time share systems (monitors such as DOS, OS, MASTER, KRONOS, etc.), and various computer hardware systems (a central processing unit, (CPU) with various peripheral controllers, CDC's 6000 series with 10 processing units subordinate to the main processing unit in the CPU, the ILLIAC IV, etc.) are further hierarchical developments.
APPLICATION TO A PARTICULAR PROGRAM SYSTEM

The Input Processor of the FLEX Processors

Appendices A and B are models, or descriptions, of two parts of the FLEX1, FLEX2 and FLEX3 computer processors in terms of the FLEX/REFLEX paradigm. The functioning of the processors is described by Overton (1973) and White and Overton (1974) essentially in holistic terms, that is, from the point of view of a user of the computer. A general mechanistic description of the simulation subprocessor of the FLEX2 and FLEX3 processors is given by White et. al. (1974). The input sub-processor and the input language definition sub-processor are the subjects of this investigation. They handle all the communication with a user (on line or batch), and so provide for the input of the information describing a model to be simulated and the conditions of simulation, such as duration and diagnostic output. Figures 6 and 7 show the relationship of the input sub-processor with the FLEX2 processor and the other participating components of the computer system. The input subprocessor and the language definition sub-processor were written almost exclusively in FORTRAN\(^1\), with only a few small portions written in COMPASS\(^2\), and are designed to operate under the OS3 operating system on a CDC 3300 computer.

\(^1\)Essentially FORTRAN IV, but with adaptations and extensions (OSU Computer Center, 1973).

\(^2\)The assembly language for the CDC 3300 (Control Data Corporation, 1968).
Equations Coded

Input Equations

Compilation (FORKAS)

Source Function Files (S.F., Y.Z., etc.)

Overlay Creation (*.12LOAD)

Function Overlay

Function Source Code Listing

Source Code Back-up

Equations Coded

Input Data

Data File Preparation (SIPS, DATs, etc.)

Command Files Creation (EDIT)

Input/Driving Data

Command Files

General Model Processor (*.12CAD)

Data Dump File (Binary)

Simulation Routine Listing (TRNSL)

Input/Driving Data Listing

Command Files Listing

Simulation Routine Back-up

Command Files Back-up

Dump File Request

Dump File Manipulation

Dump File List (*.12DFI)

Dump File Plotter (*.12PLOT)

Dump File Data Selector (*.TRANS)

Data Dump File Data

Line Printer Plots

Selected Dump File Data

CALCOMP Plots

Figure 6. Overview of the FLEX/REFLEX simulation package.
Conventions and Notation

The application of the FLEX/REFLEX paradigm to the two sub-processors necessitated the adoption of several conventions and notations. The inputs to a FORTRAN routine (PROGRAM, SUBPROGRAM, or FUNCTION) may take the form of variables in a parameter string, or the form of an external hardware device input operation (a READ statement). In the paradigm elaboration, these are identified as elements of the Z vector, and where feasible, the FORTRAN variable name used in the routine follows the Z vector index so as to permit an easy comparison of the model with the program language listing. (The FORTRAN variables are printed in capital letters.) Similarly, the program outputs take the form of either parameter string variables or an output operation (WRITE statement), and are identified by elements of the Y vector. For simplicity, the concept-
ual columns of FORTRAN arrays are denoted by letter subscripts (ie the
ISA array, Appendix C). State variables (X) are identified when they
relate to coupling variables of subsystems and when they are required to
represent internal conditions whose past values may be arguments of the
output variables Y. Additional comments are often provided to describe
the purpose or use of the input and output variables. A general explana-
tion of the purpose of each routine is provided in addition to the
behavioral conditions. This additional information is included to
increase the usefulness of the model as a documentation tool, and to
facilitate understanding by those with knowledge of FORTRAN programming.

The identified subsystems correspond to FORTRAN and COMPASS routines
and are subscripted in a manner that displays their hierarchical position.
Each system, S, is identified by appending a subscript to the subscript
string of the immediate supersystem. The appended subscript distinguishes
the system from other systems with the same immediate supersystem, and
is followed by a subscript of zero if the system itself is composed of
subsystems. Thus, the whole system under consideration is denoted by S
if it is viewed as a whole, or S₀ if it is viewed as composed of subsys-
tems. The system S₃,1,4,2 would have the hierarchical ancestry of S₀,
S₃,0, S₃,1,0, and S₃,1,4,0.

For typographical simplicity, subscripts for systems, S, and the
vectors Z, X and Y are printed following the designator. Thus S₂,3,1
and Z₁,c are printed S₂,3,1 and Z₁,c, commas being omitted when not
necessary.

In some cases a routine, such as EWRITE is identified by more than
one system identifier (S₁,1,7,1 and S₁,1,1,1,1,3). This is in recog-
nition that a particular process has been identified as a subordinate process to two or more different higher order processes. Conceptually, each identification of the process is a distinct subsystem. However, for programming efficiency, and because the language used, FORTRAN, allows for it when no prior conditions are required, the same routine is used to perform the process and is therefore described only once.

Resolution Level

As in any model of a system, a particular resolution level must be chosen. Process considered to have little or no effect on the behavior of the whole system being investigated are not considered. Processes that are completely controlled by inputs from the environment may also be excluded provided that it is understood that the model of the system is valid only under those input conditions that do not precipitate the excluded processes.

The description of the input processor is restricted to the principal routines. The implicit FORTRAN functions, such as SQRT, are not elaborated. Abnormal conditions, such as a parity error on an input-output device are not considered. Such conditions, which are unlikely to occur and in any event are beyond control of the user, are either recoverable, and thus do not affect the behavior of the input-output processes at the level modeled, or result in termination of processing.
RESULTS AND DISCUSSION

The FLEX/REFLEX paradigm quite adequately accommodates the description of the particular program system to which it was applied (See Appendices A through G). The input processor, while not the most complex of program systems, exhibits most of the characteristics of program systems. The description in Appendix A is not unlike the documentation typically provided for a program system, but by modelling the program system in the FLEX/REFLEX paradigm one has assured logical constancy and completeness. The hierarchical structure of the input processor is well identified and visible (See Appendix G) and is consistant with the structure identified in the program language.

Each routine identified as a system is fully characterized. The behavior of the system is described without reference to its internal structure and workings, but in terms of the behavior it exhibits, as defined by the outputs, under the various input conditions to which it is subject. The form, such as parameter string variable, external symbol, or output port, and the range of values of each input and output of the system is clearly described. The distinction between input and output variables is immediately apparent even when they are represented by the same FORTRAN variable. The mechanistic behavior of each system so described, such as S,1,1,7 (PGERROR), provides for the identification of the component processes (S,1,1,7,1, EWRITE and S,1,1,7,2 HARDWARE) and the associated coupling variables (X1, X2, G1, G2, Z5 and Z6). Finally, the sequential ordering of the subprocesses, when relevant such as in S,1,1,1,1,1 (REGGET), is accounted for.
The attack of certain programming problems is markedly facilitated by use of the FLEX/REFLEX paradigm. Location and correction of "bugs", program verification, alteration of the programs to accommodate changed or new specifications of behavior, and transfer of the programs from one computer system to another are several common problems to which the FLEX/REFLEX paradigm can be advantageously applied. Identification and decoupling of processes, or systems, and the isolation of certain behavior onto particular subsystem which results from the nature of hierarchical systems, permits the investigation of each subsystem individually in isolation. The subsystem onto which the deviation from specified behavior (the bug) can be completely isolated may be examined without regard to other subsystems that are not subordinate. If the bug can not be isolated onto a subsystem, no advantage is achieved, but this is usually not the case, and the advantage achieved increases with the degree to which other subsystems can be safely excluded from consideration in the search for the cause of the bug.

Parallel to the problem of debugging is the problem of program verification - assuring that the actual behavior of a program corresponds exactly to the specifications. With the behavior of each subsystem clearly defined, it becomes a much simpler task to verify each subsystem, from the most subordinate on up, than to start out by attempting to verify the behavior of the total system. Only one subsystem is verified at a time. Any bug found must result from errors in the organization of the subsystem at that system level - it can not result from bugs in any subsystems for they have already been verified.
The assistance provided by the paradigm in the transfer of a program from one computer system to another is obvious. With each subsystem of the program clearly defined, those routines that are incompatible, or would not produce the specified behavior of the new computer system are readily identified. The holistic description of the routine provides a complete, concise, and verifiable set of specifications for reprogramming.

Changed or additional specifications of behavior are easily accommodated. A comparison of the new specification of behavior with the behavioral or holistic descriptions of the subsystems immediately determines which routines must be altered or reprogrammed. The size and number of routines affected provides a good measure of the degree of effort required to meet the new specifications.

Strong economic and machine and operating system limitations dictated the use of the most efficient (in terms of development effort and operational use) programming languages and techniques for the FLEX/REFLEX processor. Convenient transferability of the processor to another computer system was sacrificed and replaced with the concept of "modular" programs, in which those processes that were most likely to vary from computer to computer, (such as S,1,1,9, HARDWARE) would be isolated in well defined and easily reprogrammed routines. This was also the reason for development of the input processor, plotting generator, simulation processors and the other FLEX/REFLEX processors shown in Figure 6. Unfortunately the programming of the FLEX processors was done before the analysis of hierarchical systems was developed to its present state. The modular approach is not inconsistent with the approach of FLEX/REFLEX
paradigm. It provided a useful framework that did indeed allow for the isolation of certain behaviors onto identified subsystems. Consideration of the problems presented in the transfer of routines from one computer system to another primarily motivated the attempts to provide explicit, well defined inputs and outputs for each routine, although insights provided by the emerging paradigm were not ignored.

The result of the attempt at concise definition of inputs and outputs and the modular approach was a group of programs and routines that are generally hierarchical systems, that when described in the framework of the FLEX/REFLEX paradigm, openly display their primary characteristics and traits. The transparency of the hierarchical systems greatly facilitates communication among those investigating the systems, such as programmers or (computer) systems analysts, by removing much of the extraneous material that surrounds most models, or descriptions, of program systems. Compare, for example, the program language model (Appendix H) with the FLEX/REFLEX model (Appendices A through G). Communication among investigators in turn permits valid criticism of the characteristics and traits of a system and comparison of the system with other systems.

For the most part, the subsystems identified exhibit well defined subprocesses. S,1,1,9, (HARDWARE), determines what kind of input-output device, such as a teletype, card reader, magnetic or paper tape reader, etc., that is associated with the logical input port (LUN). S,1,1,7, (PGERROR), provides for the printing of appropriate error messages through the selected logical output ports. The choice of immediate subprocesses into which S,1,1 (INPROC, the input processor), is broken down is not the best. Those that are identified, with the exception of S,1,1,1
(ITEMGET), are relatively minor processes. A far better choice would have been to elaborate the processes of input statement type identification, parsing of the declaratory portions of the statement, and evaluation and storage of the value, or data, portion of the statement according to statement type. This is the conceptual view of the structure of the system.

Finally, probably the most important result arising from the application of the FLEX/REFLEX paradigm to a program system is the support of the paradigm itself as a general systems paradigm. The FLEX/REFLEX paradigm, although developed as a general paradigm, was first required to be suitable for ecosystem investigation. The diversity of systems to which a general system paradigm applies measures the generality of the paradigm. The ease and effectiveness of application provide a measure of its usefulness. The successful application of the FLEX/REFLEX paradigm to the input processor adds to the evidence for the paradigm's usefulness and generality.

Relation to Top-down Structured Programming

The program structures resulting from the application of the FLEX/REFLEX paradigm to computer program systems are consistent with and similar to the structures developed in "top-down structured programming". Structured programming was first advanced by Edsger Dijsktra (1972) as a method of programming that could go beyond the limited capabilities of computer languages to deal with certain programming problems: development of programs readily understandable by
others: development of a structural format permitting comparison of one program with another designed to accomplish either the same or a different task; and particularly, development of programs, that, by the nature of their construction, could be shown to be correct. The resulting technique has been expanded and elaborated by others such as Wirth (1974) and McGowan and Kelly (1975) to yield the techniques of top-down structured programming.

Top-down structured programming emphasises the development of programs in a "stepwise" manner. The first macro level solution to the problem is successively refined in a top-down fashion until a final level is reached that is suitable for coding in the target computer language. At each level, refinement of the next higher level program elements is restricted to IF-THEN-ELSE, DO-WHILE, and statement sequencing control structures, and certain extensions, such as REPEAT-UNTIL and SELECT-CASE (McGowan, 1975). This restriction is primarily designed to enforce the one entry, one-exit nature of not only the element being refined, but also the steps constituting the refinement and thus permit continuation of the refining process. The permitted control structures were also selected so that at each level the programmer can verify that the program remains a correct formal statement. By knowing the nature of the action of the control structure used to elaborate a higher level element, and verifying that the actions of the control structures are identical with the action of the higher level element, the programmer is assured of the continued correctness of the whole program (Ledgard 1973).

Although top-down structured programming is generally viewed as taking a conceptually whole yet mechanistically simple program and pro-
gressively increasing its complexity in a systematic fashion, the structures developed are clearly hierarchical. Top-down structured programming and the FLEX/REFLEX paradigm both emphasize the initial decomposition of the "exact" statement of the problem (the holistic definition) into a mechanistic structure (the mechanistic definition of the system). The "exact" statement of the problem as understood in top-down structured programming is equivalent to the FLEX/REFLEX paradigm definition by permanent behavior. The assertion, however, that the refinements of a next higher level element properly constitute a subsystem, and that such a subsystem taken together are an hierarchical elaboration of the whole system requires some justification. The central structures to which top-down structured programming is restricted and verification of their actions against a higher level element assure the decomposition of the program into properly defined processes. The three basic constructs (IF-THEN-ELSE, DO-WHILE and statement sequencing) which were shown by Boehm and Jacopini (1966) to be sufficient for expressing any flow-chartable program logic, are of the form:

```
DO WHILE (B);
  S;
END

IF B
  THEN S₁;
  ELSE S₂;
S₁; S₂; ...
```

Where S is a sequence of program statements and B is a relation itself consisting of program statements. Since DO-WHILE and IF-THEN-ELSE are themselves program statements, the basic control structure
clearly and properly define processes in a mechanistic form. The requirement that the set of statements that decompose a next higher level element be correct necessarily implies that the element be fully defined holistically in terms of its permanent behavior.

The demonstrated compatibility and consistency of the FLEX/REFLEX paradigm with top-down structured programming is gratifying. While in no way offering any proof, it does help to reassure one that the FLEX/REFLEX approach is valid and useful.
BIBLIOGRAPHY


APPENDIX A

FLEX/REFLEX Input Processor System Description

This appendix describes the subsystems of the input processor of the FLEX/REFLEX simulation processor in the FLEX/REFLEX convention. Each subsystem is characterized according to its external quantities and their resolution levels, its permanent behavior, and its universe coupling structure. Also included is a short, imprecise description of the behavior of the subsystem in functional terms to facilitate conceptual understanding. For convenience, the mnemonic of the FORTRAN program variable is listed with the associated external variable. Appendix B provides a verbal description of the FORTRAN variables. External variables are identified with the Z vector if they are input variables and the Y vector if they are output vectors. For simplicity, subscripts are printed full size following the vector identifier.

Internal X variables are identified with Y variables when the latter is a one to one function onto the former. G variables indicate internally (for a REFLEX system) generated values.
This is the input processor itself. It evaluates one statement of the input language and processes it if syntactically correct, or, if incorrect, processes the error condition.

**EXTERNAL VARIABLES:**

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z1a</td>
<td>ISA</td>
<td>SYMBOL8</td>
</tr>
<tr>
<td>Z1b</td>
<td>ISA</td>
<td>0 ≤ INTEGER &lt; 131,069</td>
</tr>
<tr>
<td>Z1c</td>
<td>ISA</td>
<td>1 ≤</td>
</tr>
<tr>
<td>Z1d</td>
<td>ISA</td>
<td>0 ≤ INTEGER &lt; 32,768</td>
</tr>
<tr>
<td>Z1e</td>
<td>ISA</td>
<td>0 ≤ INTEGER &lt; 32,768</td>
</tr>
<tr>
<td>Z1f</td>
<td>ISA</td>
<td>0 ≤ INTEGER &lt; 32,768</td>
</tr>
<tr>
<td>Z1g</td>
<td>ISA</td>
<td>0 ≤ INTEGER &lt; 32,768</td>
</tr>
<tr>
<td>Z2</td>
<td>CBUFF</td>
<td>STRING*, N &lt; Z5</td>
</tr>
<tr>
<td>Z3</td>
<td>JC</td>
<td>1 ≤ INTEGER &lt; 141</td>
</tr>
<tr>
<td>Z4</td>
<td>JLUN</td>
<td>1 ≤ INTEGER &lt; 63</td>
</tr>
<tr>
<td>Z5</td>
<td>MAXLEN</td>
<td>1 ≤ INTEGER &lt; 137</td>
</tr>
<tr>
<td>Z6</td>
<td>MARK</td>
<td>-1 ≤ INTEGER &lt; 141</td>
</tr>
<tr>
<td>Z7</td>
<td>JOPT</td>
<td>1 ≤ INTEGER &lt; 8</td>
</tr>
<tr>
<td>Z8</td>
<td>JERR</td>
<td>-1, 0</td>
</tr>
<tr>
<td>Z9</td>
<td>CBUFF(MAXLEN+1)</td>
<td>STRING*, 0 ≤ N ≤ Z6</td>
</tr>
<tr>
<td>Z10</td>
<td>LUNA</td>
<td>0 ≤ INTEGER &lt; 63</td>
</tr>
<tr>
<td>Z11</td>
<td>LUNB</td>
<td>0 ≤ INTEGER &lt; 63</td>
</tr>
<tr>
<td>Z12</td>
<td>LUNCOPY</td>
<td>0 ≤ INTEGER &lt; 63</td>
</tr>
<tr>
<td>Z13</td>
<td>ISHIFT</td>
<td></td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y1d</td>
<td>ISA</td>
<td>0 ≤ INTEGER</td>
</tr>
<tr>
<td>Y2</td>
<td>CBUFF</td>
<td>STRING*, 1 ≤ N ≤ Z5</td>
</tr>
<tr>
<td>Y3</td>
<td>JC</td>
<td>2 ≤ INTEGER &lt; Z5 + 2</td>
</tr>
<tr>
<td>Y4</td>
<td>lun</td>
<td>STRING on logical output device Z4</td>
</tr>
<tr>
<td>Y5</td>
<td>luna</td>
<td>STRING on logical output device Z10</td>
</tr>
<tr>
<td>Y6</td>
<td>lumb</td>
<td>STRING on logical output device Z11</td>
</tr>
<tr>
<td>Y7</td>
<td>luncopy</td>
<td>STRING on logical output device Z12</td>
</tr>
</tbody>
</table>

*See Appendix D*
PERMANENT BEHAVIOR:

Definition:

The "next" statement is defined to begin with the character in Z2 (CBUFF) pointed to (as an index starting at 1) by Z3 (IC) if Z8 (JERR) = 0, otherwise, if Z8 (JERR) = -1, the next statement is defined to begin with the first character of the next symbol that is a keyword (see Appendix D) after the character pointed to by Z3 (IC). The input-character-string is continued, with editing (see Appendix E), from the input-output device identified by logical unit number, Z4 (JLUN).

I.

\[ Y7 = Y2, \text{ with insertion of } Z13 (ISHIFT) \text{ leading blanks if } Z13 (ISHIFT) \geq 0, \text{ or deletion } Z13 (ISHIFT) \text{ leading characters if } Z13 (ISHIFT) < 0. \]

If Z4 (JLUN) is a user interactive device (teletype, CRT display console, etc.), and Z6 (MARK) \( \neq 0 \), generate Y4 prior to inputing the continuation of the input-character-string from Z4 (JLUN):

\[
Y4 = \begin{cases} 
\text{Print the prompt charter "->" on Z4 (JLUN) if Z6 (MARK) = 0}, \\
\text{Print Z6 (MARK) characters of Z9 on Z4 (JLUN) if Z6 (MARK) > 1, recognizing the last two characters as a carriage-return and line-feed if they are "$\$".} 
\end{cases}
\]

In addition to those given in Appendix D (such as the start of a new statement by a valid keyword or message-request symbol), a statement
may also be terminated by any item that would be incorrect as a datum according to the parameters of Z1 (ISA) if: (1) Z4 (JLUN) is not a user interactive device (teletype, CRT, etc.); (2) Z4 (JLUN) is a user interactive device and Z7 (JOPT) = 1, 2, 3, or 4; (3) Z7 (JOPT) = 5, 6, 7, 8 and |Z1c| = 6, 7, or 8. Otherwise, the incorrect datum and the remainder of the input-character-string segment (Z2 (CBUFF)) is ignored* and the following outputs are generated:

Y5 = printed error message given by Appendix E if 1 < Z10 (LUNA) < 63 and if Z7 (JOPT) = 5 or 6,

Y6 = printed error message given by Appendix E if 1 < Z11 (LUNB) < 63 and if Z7 (JOPT) = 5 or 6,

Y7 = printed message (after Y5 and Y6 if they are printed) "REENTER VALUE" on logical unit 61.

II.

In addition to the outputs under I, the following outputs are governed by the conditions:

CØ: The next statement is syntactically correct,
C1: The next statement is syntactically incorrect,
C2: Condition CØ and the next statement is an end-of-file symbol "[][" or an end-of-file indication on logical unit Z4 (ILUN),
C3: Condition CØ and the next statement is an error message request symbol "??",
C4: Condition CØ and not Conditions C2 or C3.

Note: As explained above, if Z4 (JLUN) is a user interactive device and Z7 (JOPT) = 5, 6, 7, 8, condition C1 can not exist.

*Since the statement is not terminated, a continuation is sought from Z4 (ILUN), the user's interactive terminal. The effect then is that the user of an interactive terminal, after being informed of the error, may correct it by continuing input with a correct datum type or he may terminate the statement by continuing with a new statement.
\[
Yld = \begin{cases}
Z1d & \text{if } C1, C2, \text{ or } C3 \text{ true}, \\
Z1d + 1 & \text{if } C4 \text{ true and } |Z1c| = 4, 5, 6, 7 \text{ or } 8, \\
Z1d + m & \text{if } C4 \text{ true and } |Z1c| = 1, 2 \text{ or } 3, \\
1 & \text{if } C4 \text{ true and } |Z1c| = 9 \text{ and the datum was a SYMBOL 8}, \\
0 & \text{if } C4 \text{ true and } |Z1c| = 9 \text{ and the datum was an INTEGER},
\end{cases}
\]

Where \( m \) = number of data in the statement.

The logical input segment of the input-character-string containing the first character after the last character of the statement processed if \( C3 \) or \( C4 \) true,

\[
Y2 (CBUFF) = \begin{cases}
\text{The logical input segment of the input-character-string containing the first character after the last character of the syntactically incorrect item if } C1 \text{ true}, \\
"(EOF) " & \text{if } C2 \text{ true}.
\end{cases}
\]

Array index of first character specified for \( Y2 \) above if \( C1, C3 \) or \( C4 \) true,

\[
Y3 (JC) = \begin{cases}
Z5 + 1 & \text{if } C2 \text{ true}.
\end{cases}
\]

The error message associated with the most recent value of \( Y9 > 2 \) (see Appendix E), printed on \( Z10 \) (LUNA) if \( 1 < Z10 < 63 \), and if \( Z7 \) (JOPT) \( \neq 3 \) or if \( Z7 \) (JOPT) = 4, 6 or 8 and \( Y9 \) (IERR) \( \neq 3 \) and if \( C1 \) or \( C2 \) true,

\[
Y5 = \begin{cases}
\text{null if } C3 \text{ or } C4 \text{ true}.
\end{cases}
\]

Same as for \( Y5 \) except the message is printed on \( Z11 \) (LUNB) if \( 1 \leq Z11 \leq 63 \).
The code for the syntax error (see Appendix E; if more than one error the precedence is 13, 3, 4, 11, 5, 15, 6, 18, 7, 20, 9, 10, 12, 14, 19, 21, 22) if C1 true,

\[
Y9 (JERR) = \begin{cases} 
1 & \text{if C2, C3 or C4 true.}
\end{cases}
\]

\[
Y10 (JEOF) = \begin{cases} 
0 & \text{if C1, C3 or C4 true,} \\
1 & \text{if C2 true.}
\end{cases}
\]

The syntactically incorrect item if C1 true,

\[
Y11 (XSYM) = \begin{cases} 
\text{blanks} & \text{if C2 or C3 true} \\
Z1a & \text{if C4 true.}
\end{cases}
\]

\[
Y12 (NUM) = \begin{cases} 
0 & \text{if C1 true and } Y9 = 3 \text{ or } 4, \text{ or if C2 or C3 true,} \\
Z1h & \text{if C1 true and } Y9 = 3 \text{ or } 4, \text{ or if C4 true.}
\end{cases}
\]

\[
Y13 = \begin{cases} 
\text{null} & \text{if C1, C2 or C3 true.}
\end{cases}
\]

The "stored data" (see below) from the statement if C4 true,

\[
Y13 = \begin{cases} 
\text{null} & \text{if C1, C2 or C3 true.}
\end{cases}
\]

The "stored data" are the internal machine representations appropriate to the datum type for the FORTRAN program variable (see Appendix C) whose address is given by Z1b. If \(|Z1c| = 1, 2, 3, 4, 5, \text{ or } 6\) the data is stored as if the program variable were triply subscripted by

\[
\text{DIMENSION } X(Z1e, Z1f, Z1g)
\]

and stored according to the FORTRAN statements

\[
\begin{align*}
N &= 0 \\
\text{DO } 10 & \text{ I = I1, I2} \\
\text{DO } 10 & \text{ J = J1, J2} \\
\text{DO } 10 & \text{ K = K1, K2} \\
N &= N + 1 \\
10 & \text{ X (I, K, K) = Nth datum}
\end{align*}
\]
Where:

S1, S2 and S3 = the respective subscripts in the statement. If a subscript is explicitly present, the corresponding S takes the value of the subscript. If the subscript is implicit or not present, the corresponding S takes the value of 0.

Example:

```
KEYS (,4) = 3, 7, 8, 9
S1 = 1, S2 = 4 and S3 = 1
```

<table>
<thead>
<tr>
<th>S1 = 0</th>
<th>S1 ≠ 0</th>
<th>S2 = 0</th>
<th>S2 ≠ 0</th>
<th>S3 = 0</th>
<th>S3 ≠ 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1 = 1</td>
<td>S1</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2 = 1</td>
<td>Zle</td>
<td>S1</td>
<td>S2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J1 = 1</td>
<td></td>
<td>Zle</td>
<td>S2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K1 = 1</td>
<td></td>
<td></td>
<td></td>
<td>S3</td>
<td></td>
</tr>
<tr>
<td>K2 = 1</td>
<td></td>
<td></td>
<td></td>
<td>Zlg</td>
<td>S3</td>
</tr>
</tbody>
</table>

If |Zlc| = 8, S1 as defined previously and Zle represent physical lengths (in characters), rather than a subscript values.

Let \( L = \begin{cases} 
\min (<Zle>, S1) & \text{if } S1 \neq 0, \\
\min (<Zle>, q) & \text{if } S1 = 0. 
\end{cases} \)

Where:

q = the number of characters between the equal sign delimiter and the end of the record from the input device or the termination characters >$.

The first L data (characters in the string) are consecutively stored, four to a machine word, starting at the location specified by Zlb.

If L < Zle, n blanks are stored after the Lth datum, where:

\[ N = Zle - L. \]

A datum associated with a type 9 (|Zlc| = 9) keyword is stored as is if it is integer (and within the correct range). If the datum is a symbol, and integer (logical unit number) in the range 54 to 1 that is not already associated with an input-output device is associated (EQUIP'ped) to the OS3 file name represented by the datum, and the logical unit
number so associated is stored. (In both case the item is stored at the location whose address is given by $Z_{lb}$).

A summary of CDC3300 computer characteristics associated with $Z_{lb}$ and $Z_{lc}$ is given below. Note that for $|Z_{lc}| = 8$, $Z_{lb}$ is a word address, although the description above treats $Z_{lb}$ as if it were a character address.

| $|Z_{lc}|$ | Machine Address Form ($Z_{lb}$) | Datum Length (Words) |
|----------|----------------------------------|----------------------|
| 1        | word                             | 1                    |
| 2        | word                             | 2                    |
| 3        | character                        | $1/4$                |
| 4        | word                             | 1                    |
| 5        | word                             | 2                    |
| 6        | word                             | 2                    |
| 7        | word                             | 0                    |
| 8        | word                             | $\leq 4 \times Z_{le}$|
| 9        | word                             | 1                    |

UNIVERSE COUPLING STRUCTURE:

Subsystems:

S,1,1,1,0 ITEMGET
S,1,1,1,2 VARFIND
S,1,1,1,3 IVARGET
S,1,1,1,4 CONVERT
S,1,1,1,5 SAEQD
S,1,1,1,6 VARSTOR
S,1,1,1,7,0 PGERROR
S,1,1,1,8 HARDWARE

Couplings:

See tables 1 and 2. The outputs that are immediate functions of the internal variables are:

Y2 = X2 CBUFF,
Y3 = X3 JC,
Y4 = X4 lun,
Y5 = X5 luna,
Y6 = X6 lumb,
Y7 = X7 luncopy,
Y10 = X10 JEOFF,
### TABLE 1. S,1,1,0 Couplings - Inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>S,1,1,0</th>
<th>S,1,1,2</th>
<th>S,1,1,3</th>
<th>S,1,1,4</th>
<th>S,1,1,5</th>
<th>S,1,1,6</th>
<th>S,1,1,7,0</th>
<th>S,1,1,8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z1</td>
<td>ISA</td>
<td>Z1(ISA)</td>
<td>Z1(ISA)</td>
<td>Z1(ISA)</td>
<td>Z1(ISA)</td>
<td>Z1(ISA)</td>
<td>Z1(ISA)</td>
<td>Z1(ISA)</td>
</tr>
<tr>
<td>Z2</td>
<td>CBUFF</td>
<td>Z2 CBUFF</td>
<td>Z2 CBUFF</td>
<td>Z2 CBUFF</td>
<td>Z2 CBUFF</td>
<td>Z2 CBUFF</td>
<td>Z2 CBUFF</td>
<td>Z2 CBUFF</td>
</tr>
<tr>
<td>Z3</td>
<td>JC</td>
<td>Z3 JC</td>
<td>Z3 JC</td>
<td>Z3 JC</td>
<td>Z3 JC</td>
<td>Z3 JC</td>
<td>Z3 JC</td>
<td>Z3 JC</td>
</tr>
<tr>
<td>Z4</td>
<td>LUN</td>
<td>Z4 LUN</td>
<td>Z4 LUN</td>
<td>Z4 LUN</td>
<td>Z4 LUN</td>
<td>Z4 LUN</td>
<td>Z4 LUN</td>
<td>Z4 LUN</td>
</tr>
<tr>
<td>Z5</td>
<td>MAXLEN</td>
<td>Z5 MAXLEN</td>
<td>Z5 MAXLEN</td>
<td>Z5 MAXLEN</td>
<td>Z5 MAXLEN</td>
<td>Z5 MAXLEN</td>
<td>Z5 MAXLEN</td>
<td>Z5 MAXLEN</td>
</tr>
<tr>
<td>Z6</td>
<td>MARK</td>
<td>Z6 MARK</td>
<td>Z6 MARK</td>
<td>Z6 MARK</td>
<td>Z6 MARK</td>
<td>Z6 MARK</td>
<td>Z6 MARK</td>
<td>Z6 MARK</td>
</tr>
<tr>
<td>Z9</td>
<td></td>
<td>Z9</td>
<td>Z9</td>
<td>Z9</td>
<td>Z9</td>
<td>Z9</td>
<td>Z9</td>
<td>Z9</td>
</tr>
<tr>
<td>Z10</td>
<td>LUN</td>
<td>Z10 LUN</td>
<td>Z10 LUN</td>
<td>Z10 LUN</td>
<td>Z10 LUN</td>
<td>Z10 LUN</td>
<td>Z10 LUN</td>
<td>Z10 LUN</td>
</tr>
<tr>
<td>Z11</td>
<td>LUNB</td>
<td>Z11 LUNB</td>
<td>Z11 LUNB</td>
<td>Z11 LUNB</td>
<td>Z11 LUNB</td>
<td>Z11 LUNB</td>
<td>Z11 LUNB</td>
<td>Z11 LUNB</td>
</tr>
<tr>
<td>Z12</td>
<td>LUNCOPY</td>
<td>Z12 LUNCOPY</td>
<td>Z12 LUNCOPY</td>
<td>Z12 LUNCOPY</td>
<td>Z12 LUNCOPY</td>
<td>Z12 LUNCOPY</td>
<td>Z12 LUNCOPY</td>
<td>Z12 LUNCOPY</td>
</tr>
<tr>
<td>Z13</td>
<td>ISHIFT</td>
<td>Z13 ISHIFT</td>
<td>Z13 ISHIFT</td>
<td>Z13 ISHIFT</td>
<td>Z13 ISHIFT</td>
<td>Z13 ISHIFT</td>
<td>Z13 ISHIFT</td>
<td>Z13 ISHIFT</td>
</tr>
<tr>
<td>G1</td>
<td></td>
<td>G1</td>
<td>G1</td>
<td>G1</td>
<td>G1</td>
<td>G1</td>
<td>G1</td>
<td>G1</td>
</tr>
<tr>
<td>G2</td>
<td></td>
<td>G2</td>
<td>G2</td>
<td>G2</td>
<td>G2</td>
<td>G2</td>
<td>G2</td>
<td>G2</td>
</tr>
<tr>
<td>G3</td>
<td></td>
<td>G3</td>
<td>G3</td>
<td>G3</td>
<td>G3</td>
<td>G3</td>
<td>G3</td>
<td>G3</td>
</tr>
<tr>
<td>G5</td>
<td></td>
<td>G5</td>
<td>G5</td>
<td>G5</td>
<td>G5</td>
<td>G5</td>
<td>G5</td>
<td>G5</td>
</tr>
<tr>
<td>G7</td>
<td></td>
<td>G7</td>
<td>G7</td>
<td>G7</td>
<td>G7</td>
<td>G7</td>
<td>G7</td>
<td>G7</td>
</tr>
<tr>
<td>G8</td>
<td></td>
<td>G8</td>
<td>G8</td>
<td>G8</td>
<td>G8</td>
<td>G8</td>
<td>G8</td>
<td>G8</td>
</tr>
<tr>
<td>G9</td>
<td></td>
<td>G9</td>
<td>G9</td>
<td>G9</td>
<td>G9</td>
<td>G9</td>
<td>G9</td>
<td>G9</td>
</tr>
<tr>
<td>G10</td>
<td></td>
<td>G10</td>
<td>G10</td>
<td>G10</td>
<td>G10</td>
<td>G10</td>
<td>G10</td>
<td>G10</td>
</tr>
<tr>
<td>X1</td>
<td></td>
<td>X1</td>
<td>X1</td>
<td>X1</td>
<td>X1</td>
<td>X1</td>
<td>X1</td>
<td>X1</td>
</tr>
<tr>
<td>X14</td>
<td></td>
<td>X14</td>
<td>X14</td>
<td>X14</td>
<td>X14</td>
<td>X14</td>
<td>X14</td>
<td>X14</td>
</tr>
<tr>
<td>HARDWARE</td>
<td></td>
<td>HARDWARE</td>
<td>HARDWARE</td>
<td>HARDWARE</td>
<td>HARDWARE</td>
<td>HARDWARE</td>
<td>HARDWARE</td>
<td>HARDWARE</td>
</tr>
</tbody>
</table>

- **Z1** (ISA)
- **Z2** (CBUFF)
- **Z3** (JC)
- **Z4** (LUN)
- **Z5** (MAXLEN)
- **Z6** (MARK)
- **Z9**
- **Z10** (LUNA)
- **Z11** (LUNB)
- **Z12** (LUNCOPY)
- **Z13** (ISHIFT)
- **G1**
- **G2**
- **G3**
- **G4**
- **G5**
- **G6**
- **G7**
- **G8**
- **G9**
- **G10**
- **G11**
- **X1**
- **X14**
### TABLE 2. S,1,1,0 Couplings - Output

<table>
<thead>
<tr>
<th>X1</th>
<th>Y1 SYM</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X2</td>
<td>CBUFF</td>
<td>Y2 CBUFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X3</td>
<td>JC</td>
<td>Y3 IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X4</td>
<td>lun</td>
<td>Y8 lun</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X5</td>
<td>luna</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X6</td>
<td>lunb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X7</td>
<td>luncope</td>
<td>Y7 luncope</td>
<td>Y1 J</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X10</td>
<td>JEOF</td>
<td>Y6 EOF</td>
<td>Y1 IVARGET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Y1 luna
Y2 lunb

Y1
This routine fetches and identifies an item (SYMBOL8, INTEGER, REAL or SPCSYM) from the input-character-string.

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z1</td>
<td>ISRCHLEN</td>
<td>0 &lt; INTEGER</td>
</tr>
<tr>
<td>Z2</td>
<td>CBUFF</td>
<td>STRING*, 0 &lt; N &lt; Z5</td>
</tr>
<tr>
<td>Z3</td>
<td>IC</td>
<td>1 &lt; INTEGER &lt; 141</td>
</tr>
<tr>
<td>Z4</td>
<td>LUN</td>
<td>1 &lt; INTEGER &lt; 137</td>
</tr>
<tr>
<td>Z5</td>
<td>MAXLEN</td>
<td>1 &lt; INTEGER &lt; 141</td>
</tr>
<tr>
<td>Z6</td>
<td>MARK</td>
<td>-1 &lt; INTEGER &lt; 141</td>
</tr>
<tr>
<td>Z7</td>
<td>LUNCOPY</td>
<td>0 &lt; INTEGER &lt; 63</td>
</tr>
<tr>
<td>Z8</td>
<td>ISHIFT</td>
<td>(</td>
</tr>
<tr>
<td>Z9</td>
<td>CBUFF(MAXLEN+1)</td>
<td>STRING*, 0 &lt; N &lt; Z6</td>
</tr>
<tr>
<td>Z10</td>
<td>LUNA</td>
<td>0 &lt; INTEGER &lt; 63</td>
</tr>
<tr>
<td>Z11</td>
<td>LUNB</td>
<td>0 &lt; INTEGER &lt; 63</td>
</tr>
</tbody>
</table>

| **Output:**       |                  |                             |
| Y1                | SYM              | SYMBOL8, SPCSYM, or STRING*, 0 < N < 20 |
| Y2                | CBUFF            | STRING*, 1 < N < Z5         |
| Y3                | IC               | 2 < INTEGER < 25 + 2       |
| Y4                | ICODE            | 1, 2, 3 or 4               |
| Y5                | ICSTRT           | 1 < INTEGER < 25 + 2       |
| Y6                | IEOF             | 0 or 1                     |
| Y7                | luncopy          | STRING on logical output device Z7 |
| Y8                | lun              | STRING on logical output device Z4 |

*See Appendix D

PERMANENT BEHAVIOR:

I. If needed, the input-character string is continued, edited, according to the rules of Appendix F, from the input-output device logically specified by Z4 (LUN), and the following outputs are generated.
\[ Y_7 = Y_2, \text{ with insertion of } Z_8 \text{ (ISHIFT) leading blanks if } Z_8 \text{ (ISHIFT) } \geq 0, \text{ or deletion } <Z_8> \text{ (ISHIFT) leading characters if } Z_8 \text{ (ISHIFT) } < 0. \]

If \( Z_4 \text{ (LUN) } \) is a user interactive device (teletype, CRT display console, etc.), and \( Z_6 \text{ (MARK) } \neq 0 \), generate the output \( Y_4 \) prior to inputting the continuation of the input character string from \( Z_4 \text{ (LUN)} \):

\[
Y_4 = \begin{cases} 
\text{Print the prompt character } ">" \text{ on } Z_4 \text{ (LUN) if } Z_6 \text{ (MARK) } = 0 \\
\text{Print } Z_6 \text{ (MARK) characters of } Z_{14} \text{ on } Z_4 \text{ (LUN) if } Z_6 \text{ (MARK) } \geq 1, \text{ recognizing the last two characters as a carriage-return and line-feed if they are } "\$\$".
\end{cases}
\]

\textbf{II.}

The following outputs may be governed by the condition \( C_1 \): an end-of-file condition was detected on \( Z_4 \text{ (LUN) } \) when attempting to obtain a continuation of the character-input-string.

Let \( Q \) be the first non-blank character in \( Z_2 \text{ (CBUFF) } \) starting with the character pointed to by \( Z_3 \text{ (IC) } \).

\[
Y_1 \text{ (SYM) } = \begin{cases} 
"EOF \quad " \text{ if } C_1 \text{ true,} \\
\text{the SYMBOL8 equivalent SYMBOL if } C_1 \text{ not true and } L = \text{ALPHA,} \\
\text{a REAL or INTEGER if } C_1 \text{ not true and } L = \text{NUMERIC,} \\
L \text{ if } C_1 \text{ not true and } L = \text{SPCSYM.}
\end{cases}
\]

\[
Y_2 \text{ (CBUFF) } = \begin{cases} 
\text{The segment of the character-input-string containing the first character after the last character of the item in } Y_1 \text{ if } C_1 \text{ not true,} \\
"(EOF)" \text{ if } C_1 \text{ true.}
\end{cases}
\]
$Y3 \ (IC) = \begin{cases} 
\text{The first character specified for } Y2 \text{ if } Cl \text{ not true}, \\
Z5 + 1 \text{ if } Cl \text{ true} 
\end{cases}$

$Y4 \ (ICODE) = \begin{cases} 
1 \text{ if } Cl \text{ true or } L = \text{ALPHA}, \\
2 \text{ if } Cl \text{ not true and the item in } Y1 = \text{INTEGER}, \\
3 \text{ if } Cl \text{ not true and the item in } Y1 = \text{REAL} \\
4 \text{ if } Cl \text{ not true and } L = \text{SPCSYM} 
\end{cases}$

$Y5 \ (ICSTRT) = \begin{cases} 
\text{Position of first character in } Z2 \text{ of the item in } Y2 \text{ if } Cl \text{ not true}, \\
Z5 + 1 \text{ if } Cl \text{ true}. 
\end{cases}$

$Y6 \ (IEOF) = \begin{cases} 
0 \text{ if } Cl \text{ not true}, \\
1 \text{ if } Cl \text{ true}. 
\end{cases}$

UNIVERSE COUPLING STRUCTURE:

Subsystems:

$S,1,1,1,1,0 \ FBUMP$

Couplings:

Table 3. $S,1,1,1,0$ Couplings

<table>
<thead>
<tr>
<th>$S,1,1,1,1$ FBUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z2 CBUFF</td>
</tr>
<tr>
<td>Z3 IC</td>
</tr>
<tr>
<td>Z4 LUN</td>
</tr>
<tr>
<td>Z5 MAXLEN</td>
</tr>
<tr>
<td>Z6 MARK</td>
</tr>
<tr>
<td>Z7 LUNCOPY</td>
</tr>
<tr>
<td>Z2 CBUFF</td>
</tr>
<tr>
<td>Z3 IC</td>
</tr>
<tr>
<td>Z4 LUN</td>
</tr>
<tr>
<td>Z5 MAXLEN</td>
</tr>
<tr>
<td>Z6 MARK</td>
</tr>
<tr>
<td>Z7 LUNCOPY</td>
</tr>
</tbody>
</table>
The output variables that are immediate functions of the input variables are:

<table>
<thead>
<tr>
<th>Y2 = X2</th>
<th>Y3 = X3</th>
<th>Y6 = X1</th>
<th>Y7 = Y5</th>
<th>Y8 = Y4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBUFF,</td>
<td>IC,</td>
<td>IEOF,</td>
<td>luncopy,</td>
<td>lun.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Z8 ISHIFT</th>
<th>Z1 ISHIFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z9</td>
<td>Z8</td>
</tr>
<tr>
<td>X1</td>
<td>Y1 IEOF</td>
</tr>
<tr>
<td>X2</td>
<td>Y2 CBUFF</td>
</tr>
<tr>
<td>X3</td>
<td>Y3 IC</td>
</tr>
<tr>
<td>X4 lun</td>
<td>Y4 lun</td>
</tr>
<tr>
<td>X5 luncopy</td>
<td>Y5 luncopy</td>
</tr>
</tbody>
</table>
S,1,1,1,1,0 FBUMP

Increments the pointer, Z3, to the next character in the segment of the input-character-string currently being processed and inputs a continuation of the input string if necessary.

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td>ISHIFT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Z2</td>
<td>CBUFF</td>
<td>STRING*, 0 ≤ N ≤ Z5</td>
</tr>
<tr>
<td>Z3</td>
<td>IC</td>
<td>1 &lt; INTEGER &lt;</td>
</tr>
<tr>
<td>Z4</td>
<td>LUN</td>
<td>1 ≤ INTEGER &lt;</td>
</tr>
<tr>
<td>Z5</td>
<td>MAXLEN</td>
<td>1 ≤ INTEGER &lt; 137</td>
</tr>
<tr>
<td>Z6</td>
<td>MARK</td>
<td>-1 &lt; INTEGER &lt;</td>
</tr>
<tr>
<td>Z7</td>
<td>LUNCOPY</td>
<td>0 ≤ INTEGER &lt; 63</td>
</tr>
<tr>
<td>Z8</td>
<td>CBUFF(MAXLEN+1)</td>
<td>STRING*, 0 ≤ N ≤ 25</td>
</tr>
<tr>
<td>Output:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y1</td>
<td>IEOF</td>
<td>0 or 1</td>
</tr>
<tr>
<td>Y2</td>
<td>CBUFF</td>
<td>STRING*, 1 ≤ N ≤ Z5</td>
</tr>
<tr>
<td>Y3</td>
<td>IC</td>
<td>2 ≤ INTEGER &lt; Z5 + 2</td>
</tr>
<tr>
<td>Y4</td>
<td>lun</td>
<td>1 ≤ INTEGER &lt; 63</td>
</tr>
<tr>
<td>Y5</td>
<td>luncopy</td>
<td>0 ≤ INTEGER &lt; 63</td>
</tr>
</tbody>
</table>

PERMANENT BEHAVIOR:

The following outputs may be governed by the condition

C1: an end-of-file condition was detected on Z4 (LUN) when attempting to obtain a continuation of the character-input-string.

*See Appendix D
Y1 (IEOF) =  
\[
\begin{cases}
0 & \text{if } Z3 < Z5, \\
1 & \text{if } Z3 = Z5 \text{ and } Cl \text{ true.}
\end{cases}
\]

Y2 (CBUFF) =  
\[
\begin{cases}
Z2 & \text{if } Z3 < Z5, \\
\text{next logical segment at the input-character-string if } & \\
Z3 = Z5 \text{ and } Cl \text{ not true,} & \\
"(EOF) " & \text{if } Z3 = Z5 \text{ and } Cl \text{ true.}
\end{cases}
\]

Y3 (IC) =  
\[
\begin{cases}
1 & \text{if } Z3 = Z5 \text{ and } Cl \text{ not true}, \\
Z3 + 1 & \text{if } Z3 < Z5, \\
Z5 + 1 & \text{if } Z3 = Z5.
\end{cases}
\]

If Z3 (IC) = Z5 (MAXLEN) and if Z4 (LUN) is a user interactive device (teletype, CRT display console, etc.), and Z6 (MARK) ≠ 0, generate the output Y4 prior to inputing the continuation of the input character string from Z4 (LUN):

Y4 =  
\[
\begin{cases}
\text{Print the prompt character } ">" \text{ on Z4 (LUN) if } Z6 & \\
\text{MARK) = 0,} & \\
\text{Print } Z6 \text{ (MARK) characters of } Z \text{ on } Z4 \text{ (LUN) if } Z6 & \\
\text{MARK) } \geq 1, \text{ recognizing the last two characters as } & \\
\text{a carriage-return and line-feed if they are } "\$\$"; & \\
\end{cases}
\]

Y7 =  
\[
\begin{cases}
Z2 & \text{if } Z3 (IC) = Z5 (MAXLEN) \text{ and } Z7 (LUNCOPY) \neq 0, \\
\text{and inserting } Z1 (ISHIFT) \text{ leading blanks if } Z1 (ISHIFT) & \\
\geq 0, \text{ or deleting } Z1 (ISHIFT) \text{ leading characters if } & \\
Z8 (ISHIFT) < 0. & 
\end{cases}
\]

UNIVERSE COUPLING STRUCTURE:
Subsystems:
\[S,1,1,1,1,1,0 \text{ (RECGET)}\]
Couplings:

Table 4. S,1,1,1,1,0 Couplings

<table>
<thead>
<tr>
<th></th>
<th>S,1,1,1,1,0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECGET</td>
<td></td>
</tr>
</tbody>
</table>

| Z1  | ISHIFT      | Z1 ISHIFT |
| Z4  | LUN         | Z4 LUN    |
| Z5  | MAXLEN      | Z5 MAXLEN |
| Z6  | MARK        | Z6 MARK   |
| Z7  | LUNCOPY     | Z3 LUNCOPY|
| Z8  |             | Z2        |
| X1  | IEOF        | Y1 IEOF   |
| X2  | CBUFF       | Y2 CBUFF  |
| X3  | IC          | Y3 IC     |
| X4  | lun         | Y4 lun    |
| X5  | luncopy     | Y5 luncopy|

The output variables that are immediate functions of the internal variables are:

\[
\begin{align*}
Y1 &= X1 \quad \text{IEOF,} \\
Y2 &= X2 \quad \text{CBUFF,} \\
Y3 &= X3 \quad \text{IC,} \\
Y4 &= X4 \quad \text{lun,} \\
Y5 &= X5 \quad \text{luncopy.}
\end{align*}
\]
This routine inputs the continuation (the next segment) of the input-character-string.

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z1</td>
<td>ISHIFT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Z2</td>
<td>CBUFF(MAXLEN + 1)</td>
<td>STRING*, 0 &lt; N &lt; Z6</td>
</tr>
<tr>
<td>Z3</td>
<td>LUNCOPY</td>
<td>0 &lt; INTEGER</td>
</tr>
<tr>
<td>Z4</td>
<td>LUN</td>
<td>1 &lt; INTEGER</td>
</tr>
<tr>
<td>Z5</td>
<td>MAXLEN</td>
<td>1 &lt; INTEGER</td>
</tr>
<tr>
<td>Z6</td>
<td>MARK</td>
<td>-1 ≥ INTEGER</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y1</td>
<td>IEOF</td>
<td>0 or 1</td>
</tr>
<tr>
<td>Y2</td>
<td>CBUFF</td>
<td>STRING*, 1 &lt; N &lt; Z5</td>
</tr>
<tr>
<td>Y3</td>
<td>IC</td>
<td>2 &lt; INTEGER</td>
</tr>
<tr>
<td>Y4</td>
<td>lun</td>
<td>STRING on logical output device Z4</td>
</tr>
<tr>
<td>Y5</td>
<td>luncopy</td>
<td>STRING on logical output device Z3</td>
</tr>
</tbody>
</table>

PERMANENT BEHAVIOR:

The following outputs may be governed by the condition

Cl: an end-of-file condition was detected on Z4 (LUN).

\[
Y_1 \text{ (IEOF)} = \begin{cases} 
0 & \text{if Cl not true,} \\
1 & \text{if Cl true.} 
\end{cases}
\]

\[
Y_2 \text{ (CBUFF)} = \begin{cases} 
\text{next logical segment of the input-character-string if Cl not true,} \\
"(EOF)" & \text{if Cl true.} 
\end{cases}
\]

*See Appendix D
Y3 (IC) = \[
\begin{cases} 
1 & \text{if } Cl \text{ not true,} \\
Z5 + 1 & \text{if } Cl \text{ true.} 
\end{cases}
\]

If Z4 (LUN) is a user interactive device (teletype, CRT display console, etc.), and Z6 (MARK) ≠ 0, generate the output Y4 prior to inputing the continuation of the input-character-string from Z4 (LUN):

\[
Y4 = \begin{cases} 
\text{Print the prompt character } "\gt" \text{ on Z4 (LUN) if } Z6 \text{ (MARK)} = 0 \\
\text{Print } Z6 \text{ (MARK) characters of } Z2 \text{ on Z4 (LUN) if } Z6 \text{ (MARK)} > 1, \text{ recognizing the last two characters as a carriage-return and line-fed if they are } "\$$". 
\end{cases}
\]

Y5 = \[
\begin{cases} 
Y2 \text{ if } Z3 \text{ (LUNCOPY)} \neq 0 \text{ and inserting } Z7 \text{ (ISHIFT) leading blanks if } Z7 \text{ (ISHIFT)} > 0, \text{ or deleting } Z7 \text{ (ISHIFT) leading characters if } Z7 \text{ (ISHIFT)} < 0. 
\end{cases}
\]

UNIVERSE COUPLING STRUCTURE:

Subsystems:

S,1,1,1,1,1,1 (TTYSIGN) 
S,1,1,1,1,1,2 (STREDIT)

Couplings:

Table 5. S,1,1,1,1,1,0 Couplings

<table>
<thead>
<tr>
<th>S,1,1,1,1,1</th>
<th>S,1,1,1,1,1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z2 CBUFF</td>
<td>Z1 CBUFF</td>
</tr>
<tr>
<td>Z5 MAXLEN</td>
<td>Z2 MAXLEN</td>
</tr>
<tr>
<td>G1</td>
<td>Z1 ICHAR</td>
</tr>
<tr>
<td>X1</td>
<td>Y1</td>
</tr>
<tr>
<td>X2 CBUFF</td>
<td>Y2 CBUFF</td>
</tr>
<tr>
<td>X3</td>
<td>Y1 IST</td>
</tr>
</tbody>
</table>
The output variable that is an immediate function of an internal variable is:

\[ Y_2 = X_2 \quad \text{CBUFF}. \]
S,1,1,1,1,1,1 (TTYSIGN)

Outputs a single character or control character to the on-line user-interactive device in ASCII code.

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z1</td>
<td>ICHAR</td>
<td>BCD character</td>
</tr>
<tr>
<td>Output:</td>
<td></td>
<td>ASCII character on logical device 100</td>
</tr>
</tbody>
</table>

PERMANENT BEHAVIOR:

The BCD character, Z1 (ICHAR) is converted to ASCII code according to CDC300 conventions and outputted on device 100, which is by definition the user interactive device.
Edits the character input string according to the rules of Appendix F.

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z1</td>
<td>MAXLEN</td>
<td>1 ≤ INTEGER &lt; 137</td>
</tr>
<tr>
<td>Z2</td>
<td>CBUFF</td>
<td>STRING 1 ≤ N ≤ Z1</td>
</tr>
<tr>
<td>Output:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y1</td>
<td>IST</td>
<td>0 or 1</td>
</tr>
<tr>
<td>Y2</td>
<td>CBUFF</td>
<td>STRING 1 ≤ N ≤ Z1</td>
</tr>
</tbody>
</table>

PERMANENT BEHAVIOR:

\[
Y_1 (IST) = \begin{cases} 
1 & \text{if a "-" (left arrow) character is in } Z_2 (CBUFF) \\
0 & \text{if no "-" is in } Z_2.
\end{cases}
\]

\[
Y_2 (CBUFF) = \begin{cases} 
Z_2 (CBUFF) \text{ edited according to Appendix F, with right} \\
\text{blank fill, for the first thru } Z_1 (MAXLEN) \text{ characters, if no "-" is in } Z_2.
\end{cases}
\]

*See Appendix D
S,1,1,2 (VARFIND)

This routine locates the entry of a keyword definition entry in the array of definitions (ISA).

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zla</td>
<td>ISA</td>
<td>SYMBOL8</td>
</tr>
<tr>
<td>Z2</td>
<td>SYM</td>
<td>SYMBOL8</td>
</tr>
<tr>
<td>Output:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y1</td>
<td>J</td>
<td>0 ≤ INTEGER &lt; 32,768</td>
</tr>
</tbody>
</table>

PERMANENT BEHAVIOR:

\[
Y1 (J) = \begin{cases} 
0 & \text{if no } N \text{ exists,} \\
N & \text{if } N \text{ exists.} 
\end{cases}
\]

Where \( N \) is the index of the entry in Zla (ISA) that matches Z2.

*See Appendix D
S,1,1,2 (VARFIND)

This routine locates the entry of a keyword definition entry in the array of definitions (ISA).

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: Zlb</td>
<td>ISA</td>
<td>0 ≤ INTEGER &lt; 131,069</td>
</tr>
<tr>
<td>Output: Y1</td>
<td>IVARGET</td>
<td>INTEGER</td>
</tr>
</tbody>
</table>

PERMANENT BEHAVIOR:

Y1 (IVARGET) = value stored at the machine address Zlb.

*See Appendix D
S_1,1,4 (CONVERT)

Converts a valid string of characters to a FORTRAN program integer or real number.

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: Z1</td>
<td>SYM</td>
<td>STRING*, 1 \leq N \leq 20 or 9</td>
</tr>
<tr>
<td>Z2</td>
<td>NTYPE</td>
<td>1 or 2</td>
</tr>
<tr>
<td>Output: Y1</td>
<td>XVAR, IVAR</td>
<td>INTEGER or REAL</td>
</tr>
</tbody>
</table>

PERMANENT BEHAVIOR:

\[
Y_1 = \begin{cases} 
\text{FORTRAN integer number for } Z_1 \text{ (} N \leq 9 \text{) if } Z_2 = 2, \\
\text{FORTRAN real number for } Z_1 \text{ (} N \leq 20 \text{) if } Z_2 = 1. 
\end{cases}
\]

If Z2 (NTYPE) = 2, the FORTRAN program references IVAR; if Z2 (NTYPE) = 1, XVAR is referenced. XVAR and IVAR are FORTRAN EQUIVALENCE'd.

*See Appendix D
This COMPASS routine locates an unused logical unit number and equips it to an OS3 save-file or another logical unit number currently in use.

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>Resolution Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: Zl</td>
<td>FILE</td>
</tr>
<tr>
<td>Output: Yl</td>
<td>$0 \leq \text{INTEGER} &lt; 55$</td>
</tr>
</tbody>
</table>

PERMANENT BEHAVIOR:

$$Yl = \begin{cases} 
N & \text{where } N \text{ is the first unused logical unit number, in descending order, which is then equipped to } Zl, \text{ if an unused logical unit number exits within the range of 54 to 1, and the OS3 save file, } Zl, \text{ exists and is equipable}, \\
0 & \text{if there is no unused logical number in the range of 54 to 1 or if } Zl \text{ is unequipable (as determined by the OS3 Executive program).} 
\end{cases}$$

*See Appendix D
This COMPASS routine stores a value at an address specified by a FORTRAN variable.

**EXTERNAL VARIABLES:**

<table>
<thead>
<tr>
<th>External Variable</th>
<th>Resolution:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Form* and Range</td>
</tr>
<tr>
<td><strong>Input:</strong></td>
<td></td>
</tr>
<tr>
<td>Z1</td>
<td>0 &lt; INTEGER &lt; 131,069</td>
</tr>
<tr>
<td>Z2</td>
<td>6, 24 or 48 bit quantity</td>
</tr>
<tr>
<td>Z3</td>
<td>0, 2 or 3</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td></td>
</tr>
<tr>
<td>Y1</td>
<td>6, 24 or 48 bit quantity</td>
</tr>
</tbody>
</table>

**PERMANENT BEHAVIOR:**

\[
Y1 = \begin{cases} 
\text{the 6 bit quantity } Z2 \text{ stored at } Z1 \text{ (character address)} & \text{if } Z3 = 0, \\
\text{the 24 bit quantity } Z2 \text{ stored at } Z1 \text{ (word address)} & \text{if } Z3 = 2, \\
\text{the 48 bit quantity } Z2 \text{ stored at } Z1 \text{ (word address)} & \text{if } Z3 = 1.
\end{cases}
\]

*See Appendix D*
This routine prints error messages.

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z1</td>
<td>IERR</td>
<td>3 &lt; INTEGER &lt; 23 and not 13 or 17</td>
</tr>
<tr>
<td>Z2</td>
<td>SYM</td>
<td>SYMBOL8</td>
</tr>
<tr>
<td>Z3</td>
<td>BSYM</td>
<td>SYMBOL8</td>
</tr>
<tr>
<td>Z4</td>
<td>ICSTRT</td>
<td>1 &lt; INTEGER &lt; 137</td>
</tr>
<tr>
<td>Z5</td>
<td>LUNA</td>
<td>0 &lt; INTEGER &lt; 63</td>
</tr>
<tr>
<td>Z6</td>
<td>LUNB</td>
<td>0 &lt; INTEGER &lt; 63</td>
</tr>
</tbody>
</table>

| Output:           |                  |                             |
| Y1                | luna             | STRING on logical device Z5 |
| Y2                | lunb             | STRING on logical device Z6 |

PERMANENT BEHAVIOR:

Y5 = message if Z5 > 0.

Y6 = message if Z6 > 0.

where the message is:

line 1 "ERR ON KEYWORD" Z2 (SYM) "WHILE SCANNING" Z3 (BSYM) "AT POSITION" Z4 (ICSTRT).

line 2 message from Appendix E for the error code Z1 (IERR).

UNIVERSE COUPLING STRUCTURE:

Subsystems:

S,1,1,7,1 (HARDWARE)

*See Appendix D
Table 6. S,1,1,7,0 Couplings

<table>
<thead>
<tr>
<th>S,1,1,7,1</th>
<th>HARDWARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>Z1</td>
</tr>
<tr>
<td>X1</td>
<td>Y1</td>
</tr>
</tbody>
</table>

No output variables are direct functions of the internal variable.
This COMPASS routine determines the type of physical input-output
device equipped to a logical unit number.

**EXTERNAL VARIABLES:**

<table>
<thead>
<tr>
<th>External Variable</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: Zl</td>
<td>0 ≤ INTEGER &lt; 101</td>
</tr>
<tr>
<td>Output: Yl</td>
<td>0 ≤ INTEGER &lt; 14</td>
</tr>
</tbody>
</table>

**PERMANENT BEHAVIOR:**

<table>
<thead>
<tr>
<th>Yl</th>
<th>Device associated with Zl</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>none (Zl not equipped)</td>
</tr>
<tr>
<td>1</td>
<td>OS3 &quot;FILE&quot;</td>
</tr>
<tr>
<td>2</td>
<td>line printer</td>
</tr>
<tr>
<td>3</td>
<td>card punch</td>
</tr>
<tr>
<td>4</td>
<td>card reader</td>
</tr>
<tr>
<td>5</td>
<td>magnetic tape</td>
</tr>
<tr>
<td>6</td>
<td>teletype</td>
</tr>
<tr>
<td>7</td>
<td>plotter</td>
</tr>
<tr>
<td>7</td>
<td>OS3 &quot;NULL&quot;</td>
</tr>
<tr>
<td>8</td>
<td>CRT display console</td>
</tr>
<tr>
<td>10</td>
<td>Random access file</td>
</tr>
<tr>
<td>11</td>
<td>OS3 &quot;TASK&quot; (remote batch queue)</td>
</tr>
<tr>
<td>12</td>
<td>Mass storage file (direct access disk)</td>
</tr>
<tr>
<td>13</td>
<td>Paper tape punch</td>
</tr>
</tbody>
</table>

*See Appendix D
S,1,2,0 (VARDEF)

This routine generates and enlarges the array of definitions (ISA) used by S,1,1 (INPROC) and S,1,3 (KVARDEF).

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z1</td>
<td>ISA</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>Z2</td>
<td>VSYM</td>
<td>SYMBOL8</td>
</tr>
<tr>
<td>Z3</td>
<td>VADDR</td>
<td>0 &lt; INTEGER &lt; 131,069</td>
</tr>
<tr>
<td>Z4</td>
<td>JVTYPE</td>
<td>1 &lt; INTEGER &lt; 10</td>
</tr>
<tr>
<td>Z5</td>
<td>JVDI</td>
<td>0 &lt; INTEGER &lt; 32,768</td>
</tr>
<tr>
<td>Z6</td>
<td>JVD2</td>
<td>0 &lt; INTEGER &lt; 32,768 or SYMBOL4</td>
</tr>
<tr>
<td>Z7</td>
<td>JVD3</td>
<td>0 &lt; INTEGER &lt; 32,768 or SYMBOL4</td>
</tr>
<tr>
<td>Z8</td>
<td>NUM</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Z9</td>
<td>LUNA</td>
<td>0 &lt; INTEGER &lt; 63</td>
</tr>
<tr>
<td>Z10</td>
<td>LUNB</td>
<td>0 &lt; INTEGER &lt; 63</td>
</tr>
</tbody>
</table>

Output:            | ISA              | See Appendix C             |
| Y1                | luna             | STRING on logical device Z9 |
| Y2                | lunb             | STRING on logical device Z10 |

PERMANENT BEHAVIOR:

The following outputs may be governed by the conditions:

C1: Z7 ≠ 0 and Z5 = 0
C2: Z7 ≠ 0 and Z6 = 0
C3: Z6 ≠ 0 and Z5 = 0
C4: Z1a matches an existing entry for parameter Ea in the ISA array (See Appendix C).

*See Appendix D
Yla = Z2 (SYM)
Ylb = Z3
Ylc = Z4 (JTYPE)
Yld = 0
Yle = Z5 (JVDI)
Ylf = Z6 (JVD2)
Ylg = Z7 (JVD3)
Ylh = Z8 (NUM)

if C1, C2 and C3 all not true.

\[ Yli = \begin{cases} 
Zli + 8 & \text{if } C4 \text{ not true,} \\
Zli & \text{if } C4 \text{ true.} 
\end{cases} \]

Y2 = error message 16 (See Appendix E) if C1, C2 or C3 true and Z9 > 0.
Y3 = error message 16 (See Appendix E) if C1, C2 or C3 true and Z10 > 0.

UNIVERSE COUPLING STRUCTURE:

Subsystems:

S,1,2,1 (VARFIND) described under S,1,1,2
S,1,2,2 (PGERROR) described under S,1,1,7,0
S,1,2,3 (IGETADDR)

Couplings:

Table 7. S,1,2,0 Couplings

<table>
<thead>
<tr>
<th>Z1 ISA</th>
<th>S,1,2,1 IGETADDR</th>
<th>S,1,2,2 PGERROR</th>
<th>S,1,2,1 VARFIND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z1 ISA</td>
<td>Z1 ISA</td>
<td>Z2 SYM</td>
<td></td>
</tr>
<tr>
<td>Z9 LUNA</td>
<td>Z5 LUNA</td>
<td>Z6 LUNB</td>
<td></td>
</tr>
<tr>
<td>Z10 LUNB</td>
<td>Z1 IERR</td>
<td>Z2 SYM</td>
<td></td>
</tr>
<tr>
<td>G1</td>
<td>Z3 BSYM</td>
<td>Z4 ICSTRT</td>
<td></td>
</tr>
<tr>
<td>G2</td>
<td>Y1 IGETADDR</td>
<td>Y1 luna</td>
<td></td>
</tr>
<tr>
<td>G3</td>
<td>Y1 luna</td>
<td>Y2 lunb</td>
<td></td>
</tr>
<tr>
<td>G4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X2 luna</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X3 lunb</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X4</td>
<td></td>
<td></td>
<td>Y1</td>
</tr>
</tbody>
</table>
The output variables that are immediate functions of internal variables are:

\[ Y_2 = x_2 \text{ luna}, \]
\[ Y_3 = x_2 \text{ lunb}. \]
S,1,2,3 (IGETADDR)

This COMPASS routine retrieves the machine address of a FORTRAN call parameter and makes the address available to the FORTRAN program as a value of a FORTRAN variable.

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y1</td>
<td>IGETADDR</td>
<td>0 ≤ INTEGER &lt; 131,069</td>
</tr>
</tbody>
</table>

PERMANENT BEHAVIOR:

Y1 (IGETADDR) = machine address of Zl, a FORTRAN program variable.

*See Appendix D
This routine removes an entry from the array of definitions (ISA).

EXTERNAL VARIABLES:

<table>
<thead>
<tr>
<th>External Variable</th>
<th>FORTRAN Mnemonic</th>
<th>Resolution: Form* and Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z1</td>
<td>ISA</td>
<td>See Appendix C</td>
</tr>
<tr>
<td>Z2</td>
<td>SYM</td>
<td>SYMBOL8</td>
</tr>
<tr>
<td>Output:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y1</td>
<td>ISA</td>
<td>See Appendix C</td>
</tr>
</tbody>
</table>

PERMANENT BEHAVIOR:

\[
Y1N = \begin{cases} 
Z1n & \text{if no } Z1a \text{ entry matches } Z2, \\
Z1n-8 & \text{if a } Z1a \text{ entry matches } Z2. 
\end{cases}
\]

\[
Y1 = \begin{cases} 
Z1 & \text{if no } Z1a \text{ entry matches } Z2, \\
Z1 \text{ with the entry removed and the array 'compressed' (by moving all subsequent entries)} & \text{if a } Z1a \text{ entry matches } Z2. 
\end{cases}
\]

*See Appendix D
APPENDIX B

Description of FORTRAN Variables

BSYM Parameter passed to PGERROR holding the keyword of the syntactically incorrect statement, or the illegal item if no keyword yet identified.

CBUFF A multipurpose buffer, from which messages to an interactive user may be outputted and into which the language statements are inputted and edited. Its dimension must be four characters greater than MAXLEN, or if MARK > 4, then greater than MAXLEN + MARK. It is a character array.

CBUFF (MAXLEN + 1) This is the point in CBUFF at which the message to the interactive user begins. Such a message exists if MARK > 0 and is of length MARK.

CONVERT FORTRAN routine S,1,1,4.

FBUMP FORTRAN routine S,1,1,1,0.

HARDWARE COMPASS routine S,1,1,0.

IC The index pointer for CBUFF. It identifies the character (position 1, 2, 3 ...) currently being processed or the next to be processed.

ICCHAR The BCD character to be outputted in ASCII code to a teletype terminal by the COMPASS routine TTYSIGN.

ICODE The code for the type of item, SYM, fetched by ITEMGET.

ICSTART The position in CBUFF of the first character of the item, SYM, fetched by ITEMGET.

IEOF End-of-file indicator for the input device for the language statements.

IERR Code for syntax errors

IGETADDR COMPASS routine S,1,2,3.

INPROC FORTRAN routine S,1,1,0.

ISA See Appendix C.
ISHIFT Indicates the number of characters to be deleted from, or the number of blank characters to be added to the start of output to LUNCOPY.

ISRCHLEN The maximum number of increments to IC by ITEMGET.

IST Edit status of contents of CBUFF.

ITEMGET FORTRAN routine S,1,1,1,0.

IVAR See XVAR.

IVARGET FORTRAN routine S,1,1,3.

J Position in ISA (2, 10, 18, ...) of the entry found by VARFIND, or zero if not found.

JC See IC.

JEOF See IEOF.

JERR See IERR.

JLUN See LUN.

JOPT Specifies action to be taken when a syntax error is found.

JVD1 Specifies parameter Ee* for VARDEF.

JVD2 Specifies parameter Ef* for VARDEF.

JVD3 Specifies parameter Eg* for VARDEF.

JVTYPE Specifies parameter Ec* for VARDEF.

KVARDEF FORTRAN routine S,1,3.

LUN Logical unit number associated with the physical input device for language statements and output of user interactive terminal messages.

lun The message outputted on LUN.

LUNA A logical unit number associated with a physical output device, possibly different than the device for LUNB, for error messages.

*See Appendix C
luna The message outputted on LUNA.

LUNB A logical unit number associated with a physical output device, possibly different than the device for LUNA, for error messages.

lunb The message outputted on LUNB.

LUNCOPY The logical unit number associated with the physical output device for the copy of the input on LUN.

luncopy The output for LUNCOPY.

MARK The length, in characters, of the message to be outputted from CBUFF, or the standard prompt character print/no-print switch. The message starts at position CBUFF (MAXLEN + 1).

MAXLEN The length in characters, of the input portion of CBUFF.

NTYPE The type of conversion to be performed on SYM by CONVERT.

NUM Parameter Eh (see Appendix C) for statement processed (with keyword Ea).

PGERRLUN FORTRAN routine S,1,1,7,0.

PGERROR FORTRAN routine S,1,1,7,0.

RECGET FORTRAN routine S,1,1,1,1,1,0.

SAEQD COMPASS routine S,1,1,5.

STREDIT FORTRAN routine S,1,1,1,1,2.

SYM For the routine VARFIND, this is the keyword to be located in the ISA array. For ITEMGET, it is the item fetched from CBUFF and converted by the routine CONVERT.

TTYSIGN COMPASS routine S,1,1,1,1,1.

VADDR The FORTRAN variable in which data is usually stored that is associated with the keyword VSYM. The machine address of this variable is stored as parameter Eb (see Appendix C).

VARDEFFORTRAN routine S,1,2,0.

VARFIND FORTRAN routine S,1,1,2.

VARSTOR FORTRAN routine S,1,1,6.
**VSYM**  
The symbol to be defined as a keyword by VARDEF.

**XSYM**  
The keyword of the statement processed by INPROC.

**XVAR**  
The FORTRAN number, real or integer, of the BCD number in SYM converted by CONVERT. SVAR and IVAR are EQUIVALENCED, with XVAR being referenced for real numbers and IVAR being referenced for integer numbers.
APPENDIX C

Keyword Parameter Directory Array

The input language (see Appendix D) is in part defined by an array of parameters. This array, identified in the FORTRAN programs as the ISA array, is referred to by other portions of this documentation. The lower case identifier in these references, such as Y3b or Zle, corresponds to the descriptions of E with the same lower case identifier, such as Eb or Ee.

En = The first element of the array and elements used (eight for each entry plus one for the count) is an integer count of the number of FORTRAN integer array. Each entry defines eight parameters. The first entry occupies the second through ninth words of the array, the second occupies the tenth thru seventeenth words, and so on. The eight parameters, Ea through Eh are defined in the following manner.

Ea = Eight-character keyword identifier, <SYMBOL8> stored four characters each to the first two words of the entry, left justified with right blank fill. If a symbol has more than eight characters, it is synonymous with the eight character keyword whose characters match the first seven and last characters of the longer keyword. The longer symbol may be used in a statement in place of the equivalent keyword.

Eb = Fifteen bit memory address, right justified with left zero fill, in the third word of the entry. It is the address of the FORTRAN program variable which the FORTRAN program INPROC will use to store and reference data associated with the keyword identified by Ea. If the keyword is subscripted, data will be stored as though the FORTRAN variable were dimensioned

DIMENSION (Ee, Ef, Eg)

where Ee, Ef and Eg are defined below.

Ec = Six bit integer type code stored in the left 6 bits of the fourth word of the entry. It determines the format of the data and along with Ee, Ef, and Eg below, part of the syntax of a statement starting with the keyword identified by Ea. If Ea = 1, 2, 3, there may be one, two, three, or no subscripts in the statement, depending on Ee, Ef and Eg. If Ea = 8, there may be at most one subscript in the statement. The valid data types are:
<table>
<thead>
<tr>
<th>Ec</th>
<th>Type*</th>
<th>FORTRAN Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;INTEGER&gt;</td>
<td>One word, integer number</td>
</tr>
<tr>
<td>2</td>
<td>&lt;REAL&gt;</td>
<td>Two word, floating point number</td>
</tr>
<tr>
<td>3</td>
<td>&lt;CHAROCT&gt;</td>
<td>Character or two digit octal number</td>
</tr>
<tr>
<td>4</td>
<td>&lt;SYMBOL4&gt;</td>
<td>One word, alphanimoric variable</td>
</tr>
<tr>
<td>5</td>
<td>&lt;SYMBOL8&gt;</td>
<td>Two word, alphanumeric variable</td>
</tr>
<tr>
<td>6</td>
<td>&lt;SPCSYM&gt;</td>
<td>Two word, hollerith string ≤ 8 characters and no blanks</td>
</tr>
<tr>
<td>7</td>
<td>no datum</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>&lt;STRING&gt;</td>
<td>Integer array containing hollerith string, 4 characters per word</td>
</tr>
<tr>
<td>9</td>
<td>&lt;FILE&gt;</td>
<td>1 &lt;integer&lt; 64 or a valid OS3 file name</td>
</tr>
</tbody>
</table>

If Ec <0, there can be only one statement starting with the keyword specified by Ea, or, if more, they must be given successively in the input stream.

Ed = Sixteen bit integer, stored in the right 16 bits of the fourth word in the entry. It is a count of the number of data associated with the keyword if the type, Ec = 1 thru 6. If Ec = 7 or 8, it represents the number of times a syntactically correct statement starting with the keyword identified by Ea has been processed. If Ec = 9, it represents whether or not the datum last found associated with the keyword identified by Ea was an OS3 save file name.

Ee, Ef, Eg = Bounds for the first, second, and third subscript. Together with Eb, they define a set of memory locations where data may be stored. They also determine, along with Ec, the correct syntax for a language statement starting with the keyword in Ea (see Appendix D). Ee, Ef and Eg are stored in the fifth, sixth and seventh words of the entry as integers. If the value is non negative and less than 32,608 the value is the subscript bound. Otherwise, the value is defined by the value stored for the keyword that is equal to the value interpreted in the following manner: the CDC 330 machine representation of the value is taken as the octal representation of the first four characters of the <SYMBOL8> and expanded to a full eight characters by appending four blanks.

*See definition in Appendix D
The syntax of the statement starting with the keyword identified by Ea must be of a form consistent with the number of permitted subscripts. If Ee, Ef, and Eg are all zero, the statement may have no subscripts. If only Ee is non-zero, the statement may have at most one subscript. If only Ee and Ef are non-zero, the statement must have two subscripts. If all are non-zero the statement must have three subscripts (see Appendix D).

Eg = An integer by definition associated with the keyword identified by Ea. It occupies the eight word of the entry.
APPENDIX D

Input Language Definition

The input language processed by S,1,1 (see Appendix A), consists of statements of the form: keyword, keyword - subscripts - equal sign - data, or certain special symbol pairs. A keyword starts a statement and references a directory of parameters (Ea, Eb, ..., Eg of the ISA array described in Appendix C) which determine the acceptable statement formats and data types.

A keyword is a group of alpha-numeric characters (and * and $), beginning with an alphabetic character and having a corresponding entry in the ISA array.

The input language, processed by S,1,1,0 (INPROC), consists of statements of the form: keyword - subscripts - equal sign - data, or certain special symbols.

A keyword starts a statement and references a directory of parameters that determines the acceptable statement formats and data types. A keyword is a group of alpha numeric characters (and * and $), beginning with an alphabetic character.

\[
\begin{align*}
<\text{ALPHA}> & := \text{A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|*|$} \\
<\text{NUMERIC}> & := 0|1|2|3|4|5|6|7|8|9 \\
(5) \quad <\text{SYMBOL}> & := <\text{ALPHA}> \quad \{ \quad <\text{ALPHA}> \quad | \quad <\text{NUMERIC}> \quad } \\
<\text{SYMBOL8}> & := <\text{ALPHA}> \quad \{ \quad <\text{ALPHA}> \quad | \quad <\text{NUMERIC}> \quad } \\
<\text{KEYWORD}> & := <\text{SYMBOL8}> \text{ and is also stored in an ISA array entry as parameter Ea (see Appendix C).}
\end{align*}
\]
A keyword may also be identified in the input string by a symbol of
N characters, \((N > 8)\), in which the first seven and the Nth correspond to
the eight characters of the keyword as defined above.

Examples:

KEYS *STAR NUMBER DISTANCE

RATES STOP FORMULATE FORMULAE

Note that the last two examples are synonymous.

A subscript element is similar to the subscript notation of a
FORTRAN array, except that implicit subscripts are allowed.

\[
<\text{SUBINTGR}> := \{\text{blank}\} \{+ \} \{<\text{NUMERIC}>\}_8^{11} \text{ and } 1 \leq \text{SUBINTGR} < 32,768
\]

\[
<\text{SUB1}> := \{\text{blank}\} ( <\text{SUBINTGR}> \{\text{blank}\} )
\]

\[
<\text{SUB2}> := \{\text{blank}\} ( <\text{SUBINTGR}>, <\text{SUBINTGR}> \{\text{blank}\} )
\]

\[
<\text{SUB3}> := \{\text{blank}\} ( <\text{SUBINTGR}>, <\text{SUBINTGR}>, <\text{SUBINTGR}> \{\text{blank}\} )
\]

\[
<\text{SUBELMNT}> := <\text{SUB1}> | <\text{SUB2}> | <\text{SUB3}>
\]

Examples:

Form 1: ( ) (42)

Form 2: (, ) (8,) (,27) (11,14)

Form 3: (,,) (6,,) (4,2,) (5,,23) (15,1,11) (,4,) (,,3)

(,28,7) (,,5)

Whether or not a subscript element is permitted and the formats allowed
depend on the parameters \(E_c, E_e, E_f, \) and \(E_g\) (see Appendix C).

If \(|E_c| = 4, 5, 6, 7\) or 9, no subscript element is permitted. If
\(|E_c| = 8\) or if \(|E_c| = 1, 2, 3\) and \(E_e \neq 0 \) and \(E_f = 0\) (see description
of \(E_e\) and \(E_f\) in appendix C), then subscript form 1 is allowed. If \(|E_c| =
1, 2,\) or 3, and \(E_f \neq 0\) and \(E_g = 0\), form 2 must be present. If \(|E_c| = 1,
2, or 3 and \( E_g \neq 0 \) (and thus also \( E_e \) and \( E_f \neq 0 \)) form 3 must be present.

An equal sign delimiter must always be present unless \(|E_c| = 7\).

\[
\text{<EQSIGN>} := \{ \text{blank} \} =
\]

Example:

Data may be present, and if so, its permissible type depends on \(|E_c|\).

\[
\text{<SPCL>} := ' |(]|(]|).#|;|/|?|+|-|<>
\]

\[
\text{<CHARACTR>} := \text{<SPCL>} | \text{<ALPHA>} | \text{<NUMERIC>}
\]

(1) \[
\text{<INTEGER>} := \{ \text{blank} | \} \{ + \} \{ - \} \{ \text{<NUMERIC>} \}^2
\]

and \(-8,388,607 < \text{INTEGER} < 8,388,608\).

(2) \[
\text{<REAL>} := \{ \text{blank} | \} \text{E} \{ + \} \{ - \} \{ \text{<NUMERIC>} \}^2
\]

or \[
\{ \text{blank} | \} \{ + \} \{ - \} \{ \text{<NUMERIC>} \} \text{E} \{ + \}
\]

\[
\{ \text{<NUMERIC>} \}^2
\]

with \( n + m \leq 13 \) and not \( i = 0 \) and \( j = 0 \), and \( 0 < \text{REAL} < 2^{-1} \).

\[
\text{<OCTDIGIT>} := 0|1|2|3|4|5|6|7
\]

\[
\text{<OCTAL>} := \{ \text{<OCTDIGIT>} \}^2
\]

(3) \[
\text{<CHAROCT>} := \{ \text{blank} | \} \text{<OCTAL>} | \text{<ALPHA>}
\]

(4) \[
\text{<SYMBOL4>} := \text{<ALPHA>} \text{<ALPHA>} | \text{<NUMERIC>}^3
\]

(6) \[
\text{<SPCSYM>} := \{ \text{<CHARACTR>} \}^8
\]

(8) \[
\text{<STRING>} := \{ \text{blank} | \} | \text{<CHARACTR>}^n
\]

(9) \[
\text{<FILE>} := \text{<SUBINTGR>} \text{if less than 64} | \text{<SYMBOL8>} \text{if a valid OS3 save file name}
\]

\[
\text{<DATUM>} := \{ \text{<INTEGER>} | \text{<REAL>} | \text{<CHARACTR>} | \text{<SYMBOL4>} | \text{<SYMBOL8>} |
\]

\[
\text{<SPCSYM>} | \text{<STRING>} | \text{<FILE>}
\]

Examples:

1. 83 -10986, +47
2. E29 E-219, E + 04, 1.8 -2.7654, .72 , -.87
3. ,7., 23.4E3 ,5E-2 4.E + 06 , 11.4876E28
4. SYMB *SYMB A108
5. *A1-1% AQR -4AZ -110.4 3, a-6%%$ ZRV298AB
6. THIS IS A CHARACTER (?) STRING!

The numbers identifying the data forms correspond to the permissable data types for the same |Ec| parameter value (see Appendix C). If |Ec| = 7 no datum is allowed, and if |Ec| = 4, 5, 6, or 9 only one datum may be present. If |Ec| = 1, 2, or 3 the maximum number of data permitted is N, where:

\[ N = I \times J \times K \]

\[ I = \begin{cases} 
1 & \text{if } S1 \text{ present,} \\
\max (Ee, 1) & \text{if } S1 \text{ not present.}
\end{cases} \]

\[ J = \begin{cases} 
1 & \text{if } S2 \text{ present,} \\
\max (Ef, 1) & \text{if } S2 \text{ not present.}
\end{cases} \]

\[ K = \begin{cases} 
1 & \text{if } S3 \text{ present,} \\
\max (Eg, 1) & \text{if } S3 \text{ not present.}
\end{cases} \]

S1, S2, S3 = values of the first, second and third occurrences of the element <SUBINTGR> in <SUBELMNT>.

S1 present \( \Rightarrow \lambda > 1 \) (see definition of <SUBINTGR>.)
A special statement is defined:

\[<\text{SPSTMT}>: = \{\text{blank}\}^n \quad <[|??>\]

The last item of a statement is:

a/ the Nth datum.
b/ the keyword if \(|Ec| = 7\).
c/ the last item before another keyword.
d/ the last item before a special statement.
e/ the last item before an end-of-file condition is detected on the input-output device from which the language statements are being read.
f/ the statement terminator:

\[<\text{TERM}>: = \{\text{blank}\}^n \quad >$

APPENDIX E

Error Messages and Codes

The number in parenthesis corresponds to the error code value (IFRR) in S,1,1 (see Appendix A). The actual message printed is in quotations. Ea, Eb, Ec, Ed, Ee, Ef, Eg, refer to parameters a through g in Appendix C. Terms delimited by < > are defined in Appendix D.

(0) No error.
(1) No error.
(2) Unused.
(3) "KEYWORD NOT DEFINED". The very first item in the input-character-string or the first item after the final item of the preceding statement, is a <SYMBOL> but not a <KEYWORD>.
(4) "INVALID KEYWORD". The very first item in the input-character-string or the first item after the final item of the preceding statement is not a <SYMBOL>.
(5) "INVALID CHARACTER AFTER KEYWORD". The next item in the input-character-string is other than a correctly delimited subscript expression, <SUBELMNT>, and \( |E_c| > 1, \) \( 2, \) \( 3 \) or \( 8 \).
(6) "NON INTEGER SUBSCRIPT". An item other than a <SUBINTGR> occurs within a <SUBELMNT>.
(7) "SUBSCRIPT . GT. 32767 or ZERO". A specified subscript, <SUBINTGR>, specified within a <SUBELMNT> is greater than the limit set by Ee, Ef, or Eg for the 1st, 2nd, or 3rd subscript respectively.
(9) "VALUE MUST BE INTEGER". A datum associated with a keyword with \( |E_c| = 1 \) is not an <INTEGER>.
(10) "VALUE MUST BE REAL". A datum associated with the <KEYWORD> with \( |E_c| = 2 \) is neither an <INTEGER> or a <REAL>.
(11) "VARIABLE CANNOT BE SPECIFIED AGAIN". A statement other than the immediately preceding one was started with the same keyword as the current statement and Ec < 0.
(12) "VALUE MUST BE A SYMBOL". A datum associated with a keyword with \(|Ec| = 4\) or \(5\) is not a <SHORTSYM> or a <SYMBOL> respectively.

(13) Unused.

(14) "CHARACTER STRING TOO LONG". A character other than a blank follows a <SPCSYM> of a keyword with \(|Ec| = 6\).

(15) "EQUAL SIGN NOT FOUND". An item other than an equal sign occurs: 1) after a keyword with \(|Ec| = 4, 5, 6, 7\) or \(9\). 2) after the <SUBELMNT> when \(|Ec| = 2\) or \(3\). 3) after the <SUBELMNT> if present and \(|Ec| = 1\) or \(8\).

(16) "INVALID DEFINITION OF ARRAY BOUNDS IN VARDEF". See description of S,1,2 (VARDEF).

(17) Unused.

(18) "ANOTHER VARIABLE MUST BE SPECIFIED FIRST". Ec, Ef, or Eg for the keyword is greater than 32,767 and the keyword has not had any value stored for it (Ed = 0).

(19) "UNABLE TO EQUIP SAVED FILE". The datum for a keyword with \(|Ec| = 9\) is a <SYMBOL> but: 1) not an OS3 save file name. 2) an OS3 save file name that is "busy". 3) all logical unit numbers in the range 1 thru 54 are currently in use.

(20) "NUMBER OF SUBSCRIPTS SPECIFIED IS INVALID". The <SUBELMNT> found is invalid for the values of Ee, Ef, and Eg. The only valid combinations are:

<table>
<thead>
<tr>
<th>&lt;SUBELMNT&gt;</th>
<th>Ee</th>
<th>Ef</th>
<th>Eg</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;SUB1&gt;</td>
<td>≠ 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&lt;SUB2&gt;</td>
<td>≠ 0</td>
<td>≠ 0</td>
<td>0</td>
</tr>
<tr>
<td>&lt;SUB3&gt;</td>
<td>≠ 0</td>
<td>≠ 0</td>
<td>≠ 0</td>
</tr>
</tbody>
</table>

(21) "ILLEGAL LUN". The datum for a keyword with \(|Ec| = 9\) was an integer less than 1 or greater than 62.

(22) "DATUM MUST BE ALPHANUMERIC OR OCTAL CONSTANT .LE. 64". A datum associated with a keyword with \(|Ec| = 3\) is other than a <CHAROCT>. 
APPENDIX F

Edit Characters and Rules

Logical input records containing input language statements (see Appendix D) are edited according to the rules below. The edit characters themselves are ignored in the evaluation of the language statements. Left arrow takes highest precedence, while backslash takes lowest.

← (left arrow) Any logical record containing this character is ignored.

@ (at sign) All characters to the left of this character are ignored.

\ (backslash) The character immediately preceding (other than another backslash) is to be ignored. The effect is cumulative for multiple backslashes.

Examples (all are equivalent):

I AM NOT\UMB
I AM D@I AM NUMB
I AM NUMB
I A@ I AM NUMB
Figure 8. Algorithmic diagram of the FLEX/REFLEX input processor.
Figure 9. Conceptual diagram of the FLEX/REFLEX input processor.
APPENDIX H

Program Listing
THE FASTLINK ROUTINE HAS BEEN REMOVED AND REPLACED WITH A 
CONTINUE STATEMENT. THE ROUTINE IS NOT LOGICALLY NECESSARY,
BUT WAS USED ONLY TO IMPROVE LINKAGE EFFICIENCY.
ALSO, CALLER (OF INPROC) SUPPLIED INPUT IS NOT FUNCTIONAL,
AND ANY REFERENCE TO THE CONDITION WHERE IEOF IS .LT. 0 
SHOULD BE IGNORED.

INPROC IS A GENERAL PURPOSE INPUT PROCESSING PACKAGE WRITTEN IN 
FORTRAN (EXCEPT FOR 4 VERY SMALL SPECIAL PURPOSE ROUTINES IN ASSEM-
BLER LANGUAGE). IT IS DESIGNED TO BE CALLED BY A FORTRAN PROGRAM TO 
PROVIDE THE USER WITH EASY AND SIMPLE PROCESSING OF INPUT DATA OF 
MANY TYPES AND FORMS. IF USED FROM A STLETEYPE, IT ALLOWS FOR INTER-
ACTION AND ERROR RECOVERY. THE USER MAY INPUT (AND HAVE STORED) 
DIRECTLY INTO THE PROGRAM VARIABLE) UNSUBSCRIPTED INTEGERS AND REAL 
NUMBERS, REAL, INTEGER OR CHARACTER ARRAYS BY ROWS, COLUMNS, ELEMENTS 
OF ANY COMBINATION, FOUR AND EIGHT CHARACTER ALPHANUMERIC SYMBOLS,
EIGHT-CHARACTER OR VARIABLE LENGTH CHARACTER STRINGS, LOGICAL UNIT 
NUMBERS/FILE NAMES, AND PRESENT/NOT-PRESENT INDICATORS.

THE DATUM (DATA) FOR A FORTRAN PROGRAM VARIABLE IS IDENTIFIED BY 
A KEYWORD IN THE INPUT STRING, KEYWORDS, AND CERTAIN ATTRIBUTES,
SUCH AS DIMENSIONS, ARE DEFINED BY THE USER PRIOR TO PROCESSING THE 
INPUT FILE. THE ATTRIBUTES, TOGETHER WITH SOME ADDITIONAL PARAMETERS 
OF THE KEYWORD IN THE INPUT STRING, DESCRIBE THE DATUM (DATA) THAT 
FOLLOWS AND PROVIDE THE SPECIFICATIONS FOR BASIC VALIDITY CHECKING. 
THE PACKAGE PERMITS THE USER TO INPUT DATA SETS (IE THE DATUM (OP 
DATA) ASSOCIATED WITH A PARTICULAR KEYWORD) IN ANY SEQUENCE, AND ALSO 
ALLOWS THE USER TO FURTHER PROCESS EACH DATA SET IMMEDIATELY AFTER 
IT IS INPUT.

THE BASIC OPERATION OF THE PACKAGE IS SIMPLE. THE USER, THROUGH 
SUCCESSIVE CALLS TO SUBROUTINE VARDEF, DEFINES THE SET OF KEYWORDS TO 
BE FOUND IN THE INPUT STRING. THEN EACH SUCCESSIVE CALL TO 
SUBROUTINE INPROC CAUSES THE INPUTTING OF ONE DATA SET. ON RETURN, 
THE USER IS PROVIDED WITH INFORMATION ABOUT ANY ERRORS FOUND, IF END 
OF FILE OCCURRED, AND WHAT KEYWORD WAS JUST PROCESSED. IF THE USER 
WISHES, HE MAY THEN PROCESS THE DATA SET FURTHER, SUCH AS TO CHECK 
FOR CORRECT VALUE RANGES, BEFORE CALLING INPROC AGAIN. THE USER IS 
ALSO PROVIDED WITH CERTAIN OTHER OPTIONS AND MEANS OF MODIFYING THE 
OPERATION OF THE PACKAGE. THESE ARE EXPLAINED LATER.

A KEYWORD IS AN ALPHANUMERIC SYMBOL OF ARBITRARY LENGTH. IT MUST 
BEGIN WITH AN ALPHA CHARACTER, BUT MAY OTHERWISE BE COMPOSED OF ANY 
ALPHABETIC OR NUMERIC CHARACTER. IN THE LENGTH ENDS 8 CHARACTERS 
INPROC TRUNCATES THE SYMBOL BY TAKING THE FIRST SEVEN AND THE LAST 
CHARACTERS. THUS, SYMBOLISM IS EXACTLY EQUIVALENT TO SYMBOLISM.

THE USER DEFINES A KEYWORD BY A CALL TO VARDEF AS FOLLOWS.
CALL VARDEF(IARRAY,XKEYWORD,VAR,ITYPE,IOIM,IOIL,IOILS,NUM)
WHERE
IARRAY=THE INTEGER ARRAY THAT HOLDS THE DEFINITIONS. ITS MINIMUM
LENGTH MUST BE 8 TIMES THE NUMBER OF KEYWORDS PLUS 1. THE
FIRST WORD OF THE ARRAY IS USED AS A POINTER TO THE END OF THE
DEFINITION LIST AND MUST BE ZERO BEFORE THE FIRST CALL TO
VAPDEF. RESETTING IT TO ZERO EFFECTIVELY WIPES OUT THE LIST
OF DEFINITIONS.

XKEYWORD= TWO-WORD (REAL) SYMBOL KEYWORD. EXAMPLES,
MARY
MARYJANE

THE LATTER KEYWORD IS EQUIVALENT TO MARYJANICE OR MARYJANXXXX,
NEITHER OF WHICH COULD BE GIVEN IN THE INPUT RECORD.
VAR=THE INTEGER OR REAL PROGRAM VARIABLE WHERE THE DATUM IS TO BE
STORED. IF SUBSCRIPTED DATA, IT IS THE STARTING LOCATION FOR
STORAGE. EXAMPLES,

ARY(1)
ARY(20,1)

ITYPE= THE TYPE OF VARIABLE, INTEGER, REAL, ETC. CODES ARE GIVEN
BELOW.
ITDIM1, ITDIM2, ITDIM3 = DIMENSIONS. IF ALL ARE ZERO, THE VARIABLE IS
UNSUBSCRIBED. THE DIMENSIONS TO BE GIVEN ARE THE SAME AS ONE
WOULD SPECIFY IN A FORTRAN DIMENSION STATEMENT. A VALUE OF
ZERO INDICATES NO DIMENSION.
NUM=ANY INTEGER THE USER WISHES TO ASSOCIATE WITH THE KEYWORD.
WHEN THE KEYWORD IS FOUND IN THE INPUT STRING, THIS INTEGER IS
ALSO AVAILABLE TO THE USER.

CODES FOR ITYPE:
1= INTEGER VARIABLE. DATA MUST BE INTEGER NUMBERS.
2= REAL VARIABLE. DATA MAY BE GIVEN AS INTEGER OR REAL NUMBERS
IN THE INPUT STRING. INTEGERS ARE CONVERGED TO FLOATING
POINT (REAL) NUMBERS BEFORE STORING.
3= CHARACTER VARIABLE. DATA MUST BE SINGLE ALPHANUMERIC
CHARACTERS OR TWO DIGIT OCTAL CODES (LE. 34). EXAMPLES.

  D
  9
51 (OCTAL CODE FOR D)
11 (OCTAL CODE FOR 9)

4= INTEGER SYMBOL VARIABLE. THE FIRST FOUR CHARACTERS OF THE
SYMBOL FOLLOWING THE DELIMITING EQUAL SIGN (SYMBOL DEFINED
IN THE SAME WAY AS A KEYWORD) IS STORED IN THE VARIABLE.
5= SAME AS 4 EXCEPT THE WHOLE SYMBOL (REDUCED TO 8 CHARACTERS
IF NECESSARY) IS STORED IN THE REAL VARIABLE.
6= TWO-WORD SPECIAL CHARACTER VARIABLES. THE CHARACTERS
FOLLOWING THE DELIMITING EQUAL SIGN, UNTIL A BLANK IS FOUND, ARE STORED LEFT JUSTIFIED IN THE VARIABLE WITH BLANK FILL. ANY CHARACTER EXCEPT \, (BACK SLASH), \ (AT SIGN), OR \ (LEFT ARROW) IS VALID.

\(7=\text{PRESENT}/\text{NOT PRESENT INDICATOR. NO DATA IS STORED. INPROC RETURNS IMMEDIATELY AFTER FINDING THE KEYWORD. ONLY INDICATING ITS PRESENCE IN THE INPUT STRING. THIS IS USEFUL FOR COMMANDS IN A USER INTERACTIVE PROGRAM.}\)

\(8=\text{VARIABLE LENGTH ITEM. THIS IS SIMILAR TO 3 EXCEPT THAT IT HAS A DIFFERENT INPUT FORM (SEE BELOW) AND PROVIDES FOR BLANK FILL. AND MUST BE AN INTEGER OR REAL VARIABLE. A TYPE 8 VARIABLE MUST BE SIMPLY DIMENSIONED (IDIM1 NOT EQUAL 0). CHARACTERS ARE FETCHED AND STORED UNTIL ONE OF THE FOLLOWING IS MET. A) THE END OF THE PHYSICAL INPUT RECORD. B) THE NUMBER SPECIFIED BY SUBSCRIPT HAS BEEN Fetched. C) THE NUMBER SPECIFIED BY IDIM1 HAS BEEN FETCHED. IF A) OR B) IS MET, THE VARIABLE ARRAY IS BLANK FILLED UP TO IDIM1 (IDIM1 SPECIFIES THE MAXIMUM NUMBER OF CHARACTERS IN THE REAL OR INTEGER ARRAY.}\)

\(9=\text{LOGICAL UNIT NUMBER/FILE NAME. THE VARIABLE MUST BE TYPE INTEGER. IF THE DATUM IS A SYMBOL, IT IS ASSUMED TO BE A FILE NAME AND AN UNUSED LUN (FROM 54 TO 1) IS EQUIPPED TO THE FILE. THE VALUE OF THE LUN IS THEN STORED IN THE VARIABLE IF THE VARIABLE HAS THE VALUE OF A LUN PREVIOUSLY EQUIPPED BY INPROC, THAT LUN IS FIRST UNEQUIPPED. IF A SLASH AND A SYMBOL BEGINNING WITH P IMMEDIATELY FOLLOW THE LUN OR FILE NAME, THE LUN IS REWOUND.}\)

**INPUT FORMS**

**UNSUBSCRIPTED ITEMS ARE OF THE FORM KEYWORD=DATUM**

**EXAMPLES**

\( \text{KIL0=}20.5 \quad \text{TYPE} \ 2 \)

\( \text{SYM=}\text{ABCDEFGHIJKLMNOPQRSTUVWXYZ} \quad \text{TYPE} \ 5 \)

\( \text{INLUN=}\text{DATAFILE/REWINO} \quad \text{TYPE} \ 9 \)

\( \text{MTSC=}A.B \quad \text{TYPE} \ 6 \)

\( \text{PRINT} \quad \text{TYPE} \ 7 \)

\( \text{OUTLUN=}20/\text{REW} \quad \text{TYPE} \ 9 \)

\( \text{CHAR=}A \quad \text{TYPE} \ 3 \)

\( \text{N=}5 \quad \text{TYPE} \ 1 \)

\( \text{ICYM=}021XY2 \quad \text{TYPE} \ 5 \quad (\text{ONLY 021X WOULD BE STORED}) \)

\( \text{MISCCL=}A(21) \quad \text{TYPE} \ 8 \)

**FORMS FOR SUBSCRIPTED VARIABLES ARE AS FOLLOWS**

\( \text{AR()}=\text{DATA} \)

\( \text{LIST(1)}=\text{DATA BY ROWS} \)

\( \text{LIST(4)}=\text{DATA FOR 4TH ROW} \)
LIST(2,3)=DATA FOR 3RD COLUMN

LIST(2,2)=DATUM

DOO(),(),=DATA BY ROWS, BY COLUMNS, BY DEPTH.

THE ROW, COLUMN, AND DEPTH DIMENSIONS ARE GIVEN BY IDIM1, IDIM2, AND IDIM3.

IF A KEYWORD IS FOUND BEFORE ALL DATA IS Fetched, INPROC WILL ASSUME THAT IS THE END OF THE DATA. Thus, if KEYWORD LOID IS DEFINED WITH IDIM1=5, AND LAD IS ALSO A KEYWORD, AND LID()=20, 1, 2, 3, LAD=... TS INPUT, INPROC WILL STORE ONLY THE 3 NUMBERS GIVEN, AND RETURN. ON THE NEXT CALL TO INPROC, LAD WILL BE PROCESSED.

THE FORM FOR TYPE B VARIABLES IS
HERB=21 CHARACTERS

REAL NUMBERS ARE OF THE FOLLOWING FORMS

23.1
3.2e+10
-2.764E12

NUMBERS ARE SEPARATED BY A BLANK OR COMMA.
SPACES MAY NOT BE EMBEDDED IN SYMBOLS OR NUMBERS, BUT ARE OTHERWISE VALID ANYWHERE IN THE INPUT STRING. A DATUM MUST BE COMPLETE ON A PHYSICAL INPUT RECORD (E.g., THE NUMBER 200, 867F-32 COULD NOT START IN COLUMN 76 OF A CARD AND BE CONTINUED ON THE NEXT). DATA FOR AN ARRAY, HOWEVER, CAN BE CONTINUED ON SUCCESSIVE RECORDS. (EXAMPLE, FOR THE DIMENSION 5 ARRAY, HERB()=20.0 4.8 1 87.1 .06, EACH NUMBER COULD BE ON A SEPARATE RECORD.)

NEGATIVE AND ZERO SUBSCRIPTS ARE INVALID, AND SUBSCRIPTS MAY NOT EXCEED THE CORRESPONDING DIMENSION IN THE VARIETY CALL (IDIM1, IDIM2, IDIM3).

THE FOLLOWING EDIT SYMBOLS ARE AVAILABLE. WHEN PRESENT IN THE INPUT DATA, THEIR EFFECT IS AS FOLLOWS.
  • (BACKSLASH) DELETE IMMEDIATELY PRECEDEING CHARACTER, THE EFFECT IS CUMULATIVE. DOUBLE BACKSLASHES WILL DELETE PREVIOUS TWO CHARACTERS. EXAMPLES, TXX\ THIS IS IT.
  • (AT SIGN) ALL CHARACTERS TO LEFT (IN PHYSICAL RECORD) ARE IGNORED. EXAMPLE, THX\ THIS IS IT IS READ AS THIS IS IT.
  • (LEFT ARROW) THE ENTIRE RECORD IS IGNORED AND THE NEXT RECORD IS READ.

EACH CALL TO THE SUBROUTINE INPROC PROCESSES THE NEXT KEYWORD AND ITS DATUM (DATA).

CALL INPROC(ARRAY, BUFFER, LUN, IEOF, INDEX, BUPPERLEN, WORDKEY, NUM, TERR, IOPT, MARK)
ARRAY = THE KEYWORD DEFINITION ARRAY, THE SAME ARRAY AS IN THE VARDEF PARAMETER STRING.

CBUFFER = CHARACTER TYPE ARRAY OF LENGTH 140. THIS IS USED AS THE INPUT/OUPUT BUFFER.

LUN = THE INPUT/OUTPUT BUFFER.

EOF = END OF FILE INDICATOR. 1=EOF, 0=NOT EOF.

INDEX = CHARACTER INDEX OF CBUFFER. IF THE USER WISHES TO READ A RECORD BEFORE STARTING, INDEX SHOULD BE INITIALLY SET TO A VALUE GREATER THAN 140. OTHERWISE IT SHOULD BE SET TO 1.

WORDKEY = THE TWO WORD VARIABLE CONTAINING THE LEFT-JUSTIFIED (WITH BLANK FILL) KEYWORD JUST PROCESSED BY INPROC.

LUN = THE INPUT FILE LUN. MUST BE IN THE RANGE OF 1 TO 60.

IEOF = END OF FILE INDICATOR. 1=EOF, 0=NOT EOF.

INDEX = CHARACTER INDEX OF CBUFFER. IF THE USER WISHES TO READ A RECORDD BEFORE STARTING, INDEX SHOULD BE INITIALLY SET TO A VALUE GREATER THAN 140. OTHERWISE IT SHOULD BE SET TO 1.

WORDKEY = THE TWO WORD VARIABLE CONTAINING THE LEFT-JUSTIFIED (WITH BLANK FILL) KEYWORD JUST PROCESSED BY INPROC.

LUN = THE INPUT FILE LUN. MUST BE IN THE RANGE OF 1 TO 60.

IEOF = END OF FILE INDICATOR. 1=EOF, 0=NOT EOF.

INDEX = CHARACTER INDEX OF CBUFFER. IF THE USER WISHES TO READ A RECORDD BEFORE STARTING, INDEX SHOULD BE INITIALLY SET TO A VALUE GREATER THAN 140. OTHERWISE IT SHOULD BE SET TO 1.

WORDKEY = THE TWO WORD VARIABLE CONTAINING THE LEFT-JUSTIFIED (WITH BLANK FILL) KEYWORD JUST PROCESSED BY INPROC.

NUM = THE INTEGER ASSIGNED TO THE KEYWORD BY THE USER IN THE CALL TO VARDEF.

IERR = ERROR CODE. 1=NO ERROR. (SEE BELOW FOR LIST OF ERRORS)

IOPT = OPTION CODE.

1 = IF ERROR OCCURS, PRINT ERROR MESSAGE AND ABORT.

2 = IF ERROR OCCURS, PRINT ERROR MESSAGE AND RETURN. INDEX WILL POINT TO THE NEXT ITEM (POSSIBLY WITH LEADING BLANKS) IN CBUFFER AFTER THE ITEM THAT WAS IN ERROR. IERR WILL CONTAIN THE ERROR CODE.

3 = SAME AS 2 EXCEPT NO ERROR MESSAGE IS PRINTED.

4 = SAME AS 2 EXCEPT THAT WHEN AN UNDEFINED KEYWORD IS FOUND (IE, A SYMBOL IS FOUND WHEN A KEYWORD IS EXPECTED), NO ERROR MESSAGE IS PRINTED. IF IERR=3 IF UNDEFINED SYMBOL IS FOUND.

5 = SAME AS 2 EXCEPT IF THE USER IS AT A TTY, INPROC MAKES THE USER CORRECT THE ERROR BEFORE RETURNING TO THE CALLING PROGRAM.

6 = SAME AS 5 EXCEPT IF A SYMBOL NOT DEFINED AS A KEYWORD IS FOUND WHEN A KEYWORD IS EXPECTED, INPROC WILL PRINT ONLY A SHORT ERROR MESSAGE WHEN AN ERROR IS FOUND (ENTERED OR RENTER VALUE), ENTRY OF DOUBLE QUESTION MARK (??) WILL CAUSE THE PRINTING OF THE COMPLETE MESSAGE.

MARK = (EFFECTIVE ONLY WHEN INPUT IS FROM TTY) PROMPT CODE.

1 = IF T-T. 0, NO PROMPT CHARACTER IS PRINTED. IF 0, THE STANDARD PROMPT CHARACTER, > (.GT. CHARACTER) IS PRINTED.

2 = SAME AS 1 AND 8 = SAME AS 5 AND 6 = EXCEPT, IF INPUT IS FROM TTY INPROC WILL PRINT ONLY A SHORT ERROR MESSAGE WHEN AN ERROR IS FOUND (ENTERED OR RENTER VALUE). ENTRY OF DOUBLE QUESTION MARK (??) WILL CAUSE THE PRINTING OF THE COMPLETE MESSAGE.

MARK = (EFFECTIVE ONLY WHEN INPUT IS FROM TTY) PROMPT CODE.

1 = IF T-T. 0, NO PROMPT CHARACTER IS PRINTED. IF 0, THE STANDARD PROMPT CHARACTER, > (.GT. CHARACTER) IS PRINTED.

2 = SAME AS 1 AND 8 = SAME AS 5 AND 6 = EXCEPT, IF INPUT IS FROM TTY INPROC WILL PRINT ONLY A SHORT ERROR MESSAGE WHEN AN ERROR IS FOUND (ENTERED OR RENTER VALUE). ENTRY OF DOUBLE QUESTION MARK (??) WILL CAUSE THE PRINTING OF THE COMPLETE MESSAGE.

MARK = (EFFECTIVE ONLY WHEN INPUT IS FROM TTY) PROMPT CODE.

1 = IF T-T. 0, NO PROMPT CHARACTER IS PRINTED. IF 0, THE STANDARD PROMPT CHARACTER, > (.GT. CHARACTER) IS PRINTED.

2 = SAME AS 1 AND 8 = SAME AS 5 AND 6 = EXCEPT, IF INPUT IS FROM TTY INPROC WILL PRINT ONLY A SHORT ERROR MESSAGE WHEN AN ERROR IS FOUND (ENTERED OR RENTER VALUE). ENTRY OF DOUBLE QUESTION MARK (??) WILL CAUSE THE PRINTING OF THE COMPLETE MESSAGE.
CHARACTER.

SPECIAL INDICATORS-
(1) (DOUBLE LEFT Bracket) The user may place this at the end of
the input string to indicate end of file. This will cause
INPROC to return with IEOF=1.

> & (GREATER THAN SIGN AND DOLLAR SIGN) These two characters
terminate a data set. This is particularly useful for
marking the end of type 3 variable when a subscript is not
specified (or even if it is).

**= (DOUBLE QUESTION MARK) This character pair causes the
printing of the complete error message for the last error
detected. Useful when IOPT set to 7 and 8 (see WRITE UP
FOR SUBROUTINE INPROC).

(EOF) - This internally generated symbol indicates that an
end-of-file condition has been found on the input file.
It may appear in an error message if processing of the last
keyword was not completed.

Errors are printed on LUN 61 unless the user specifies otherwise
through a call to PGE2RLUN-
CALL PGE2RLUN(LUN1,LUN2,DUMMY,DUMMY)
WHERE
LUN1= ERROR MESSAGE LUN.
LUN2= 2 NO ERROR MESSAGE LUN. IF A NON-ZERO VALUE IS SPECIFIED, AND
IS NOT THE SAME AS LUN1, ERROR MESSAGES WILL ALSO BE PRINTED ON
THIS LUN.
DUMMY= A 3RD AND 4TH PARAMETER ARE OF NO SIGNIFICANCE BUT MUST BE
PRESENT TO PROVIDE FOR LINKAGE.

A counter of each time a keyword is processed (except for type 9
variables) is kept in the definition array. It is the last 3 chars
of the 4th word in the array entry. The keyword (stored in the 1st
two words of the entry) can be located by a call to VARFIND-
CALL VARFIND(IARRAY, WORDKEY, IDX)
WHERE
IARRAY= DEFINITION ARRAY
WORDKEY= A CHARACTER KEYWORD TO BE SEARCHED FOR.
IDX= INDEX OF THE FIRST WORD OF THE ENTRY. (ZERO IF KEYWORD NOT
FOUND) THE COUNTER WORD IS THE IARRAY(IDX+3) AND MAY BE
USED BY ANADING WITH THE LITERAL =4H0000. EXAMPLE,
ICOUNT= AND(IARRAY(IDX+3),4H0000)
(THE FIRST CHARACTER CONTAINS THE KEYWORD TYPE CODE.)

A copy of each input record read by INPROC will be made if
THE USER EXECUTES A CALL TO INPTCOPY.
CALL INPTCOPY(DUMMY, LUN, ISHIFT, DUMMY, DUMMY, DUMMY)

WHERE:
LUN=OUTPUT LUN FOR THE COPY OF THE INPUT RECORD.
ISHIFT=CHARACTER SHIFT ON OUTPUT (VALUE OF 2 CAUSES INSERTION OF 2 BLANKS IN FRONT OF THE COPY, -2 WOULD CAUSE DELETION OF FIRST TWO CHARACTERS.
DUMMY=DUMMY PARAMETER TO SATISFY LINKAGE REQUIREMENTS.

THE USER MAY SPECIFY THAT THE DEFINING DIMENSION OF A SUBSCRIPTED VARIABLE BE DEFINED BY ANOTHER INPUT VARIABLE. FOR EXAMPLE, GIVEN THE TWO DEFINING CALLS-

CALL VARDEF(IARRAY,5Hx ,NN,1,0,0,0,0)
CALL VARDEF(IARRAY,5Hx ,x,2,4HN,0,0,0)

WHEN THE KEYWORD X IS FOUND IN THE INPUT STRING, INPROC WILL SEARCH THE DEFINITION ARRAY FOR THE KEYWORD N. IF NO VALUE HAS BEEN STORED IN VARIABLE NN (KEYWORD N NOT YET SPECIFIED) AN ERROR SITUATION OCCURS (SEE IOPTION PARAMETER IN INPROC CALL). OTHERWISE THE VALUE OF NN IS USED AS THE DIMENSION SPECIFICATION. INPROC WOULD ACCEPT DATA FOR X(1),...,X(N). X(N+1) WOULD NOT BE ACCEPTED.

THE USER MAY ALSO SPECIFY THAT A KEYWORD BE GIVEN ONLY ONCE (SUCCESSIVE OCCURRENCES OF THE SAME KEYWORD ARE ALLOWED). THIS PERMITS THE USER ENTERING INPUT AT LEAST A QUICK CHANGE OF HEART). THIS IS DONE BY SPECIFYING A NEGATIVE VARIABLE TYPE CODE IN THE VARDEF CALL (EXAMPLE, -2 INSTEAD OF 2). INPROC EXAMINES THE COUNTER WORD IN THE DEFINITION ARRAY. IT MUST BE ZERO OR THE PREVIOUS KEYWORD MUST HAVE BEEN THE SAME AS THE CURRENT ONE FOR INPROC TO ACCEPT DATA FOR THE KEYWORD'S VARIABLE. THIS IS PARTICULARLY USEFUL WHEN DEFINING ARRAY BOUNDS. EXAMPLE-

CALL VARDEF(IARRAY,5Hx ,inx,-1,0,0,0,0)
CALL VARDEF(IARRAY,5Hx ,jnx,-1,0,0,0,0)

DATA CANNOT BE SPECIFIED FOR ARRAY XY UNTIL DATA HAS BEEN SPECIFIED FOR I AND J. SINCE I AND J CAN BE SPECIFIED ONLY ONCE, THE DIMENSIONS OF XY ARE FIXED. ELEMENT XY(3,2) WILL ALWAYS BE STORED IN THE SAME LOCATION IN VARIABLE ARRAY XY.

IF THE USER WISHES TO DELETE A KEYWORD FROM THE DEFINITION ARRAY, HE CAN SEARCH FOR THE WORD AND BLANK IT OUT (CALL VARDEF(IARRAY, WORDKEY,J) AND SET IARRAY(J)=4H), OR USE THE SUBROUTINE VARKILL

CALL VARKILL(IARRAY,WORDKEY)

WHERE
IARRAY=KEYWORD DEFINITION ARRAY
WORDKEY=KEYWORD WHOSE DEFINITION IS TO BE DELETED.
VAPKILL is more time consuming than the first technique, but it saves space since it compresses the definition array after deleting the keyword.

The input file normally is read by INPROC. However, the user may use his own routine to do this:

1. Specify a LUN greater than 100 in the INPROC call,
2. Fill the character I/O buffer, CBUFFER, with the input record (LENGTH MAX_LFN and the user is responsible for supplying trailing blanks),
3. Set INDEX to point to the first character in CBUFFER (usually 1).

When more input is required, INPROC will return with IEOF=-1. The user must refill CBUFFER, reset INDEX, and return (without altering IEOF) directly to INPROC.

All error messages from INPROC are printed by subroutine EWRITE. The call is-

\[ \text{CALL EWRITE(LUN,LENGTH,IBUFFER)} \]

where

- LUN=OUTPUT LUN
- LENGTH=LENGTH IN WORDS OF MESSAGE
- IBUFFER=WORD ARRAY HOLDING BCD MESSAGE

If the user wishes, he may replace the binary deck of the EWRITE subroutine with his own deck.

The user also may use the item fetching routine, ITEMGET. This routine searches CBUFFER, starting at the character pointed to by INDEX and ignoring preceding blanks, and returns with the next item - a symbol, integer number, real number, special character, and error (only blanks found). Items are all in BCD, left justified with blank fill. The call is-

\[ \text{CALL ITEMGET(CBUFFER,LUN,IEOF,INDEX,SYMBOL,BUFFLEN,ICSTART,ICODE,ISRCHLEN,MARK)} \]

where

- CBUFFER=I/O BUFFER (CHARACTER ARRAY)
- LUN=INPUT LUN
- IEOF=END OF FILE INDICATOR
- INDEX=_POINTER FOR CBUFFER
- SYMBOL=DOUBLE WORD ARRAY OF DIMENSION 3 CONTAINING THE ITEM FOUND (IN BCD)
- BUFFLEN=LENGTH OF INPUT BUFFER (CHARACTER)
- ICSTART=VALUE OF IC AT 1ST NON BLANK CHARACTER FOUND IN BUFFER
- ICODE=TYPE OF ITEM FOUND
  - 1=ALPHANUMERIC SYMBOL
  - 2=INTEGER
ISRCHLEN = NUMBER OF CHARACTERS TO BE EXAMINED IN BUFFER (ROUTINE STOPS AT POSITION MAXLEN), AND CALLS FOR MORE INPUT.
MARK = PARAMETER AS DEFINED FOR INPROC.
THE FIRST FOUR PARAMETERS ARE THE SAME AS THOSE IN THE INPROC CALL.
EXAMPLE - FOR THE INPUT STRING
THIS STRING, HAS 23, 47.4E-10
ITEMGET WILL RETURN WITH (STRING) (HAS) (23) (.) (47.4E-10) AND (I)
WHERE THE CONTENTS OF SYMBOL ARE WITHIN THE PARENTHESES AFTER EACH CALL TO ITEMGET.
ERROR CODES -
1 = NO ERROR
2 = UNUSED
3 = SYMBOL NOT A KEYWORD, INPROC HAD COMPLETED PROCESSING A KEYWORD AND WAS EXPECTING THE NEXT KEYWORD.
4 = SYMBOL NOT FOUND, INPROC HAD COMPLETED PROCESSING A KEYWORD AND WAS EXPECTING THE NEXT KEYWORD. SOMETHING OTHER THAN A SYMBOL WAS FOUND.
5 = INVALID CHARACTER FOLLOWED KEYWORD.
6 = NON-NUMERIC SUBSCRIPT FOLLOWING KEYWORD FOR A DIMENSIONED VARIABLE.
7 = SUBSCRIPT FOLLOWING KEYWORD GREATER THAN 32767 OR LESS THAN OR EQUAL TO ZERO.
8 = SUBSCRIPT EXCEEDS MAXIMUM SPECIFIED FOR IT IN CALL TO VARDEF (IDIM1, IDIM2, OR IDIM3)
9 = DATUM INVALID - MUST BE INTEGER.
10 = DATUM INVALID - MUST BE INTEGER OR REAL
11 = KEYWORD MAY NOT BE SPECIFIED MORE THAN ONCE AND HAS ALREADY BEEN SPECIFIED.
12 = DATUM INVALID - MUST BE SYMBOL.
13 = END OF FILE BEFORE KEYWORD PROCESSING COMPLETED - SUCH AS END OF FILE WHEN ONLY THE KEYWORD OF A VARIABLE REQUIRING DATA WAS THE LAST ITEM IN THE INPUT STRING.
14 = MORE THAN 3 CHARACTERS OF DATA FOR A TYPE 6 VARIABLE.
15 = NO EQUAL SIGN AFTER KEYWORD.
16 = IDIM1, IDIM2, OR IDIM3 NOT CORRECTLY SPECIFIED IN CALL TO VARDEF.
17 = UNUSED
18 = A DIMENSION OF THE VARIABLE IS SPECIFIED BY ANOTHER VARIABLE, ALSO DEFINED BY A KEYWORD, AND WHICH HAS NOT YET HAD A VALUE STORED IN IT. THUS THE ARRAY CURRENTLY BEING PROCESSED HAS AN
19 = INPROC WAS UNABLE TO EQUIP THE FILE NAME (DID NOT EXIST, BUSY, 
etc) ON A TYPE 9 VARIABLE.

20 = TOO FEW OR TOO MANY SUBSCRIPTS SPECIFIED FOR VARIABLE.

21 = LUN SPECIFIED NOT IN THE RANGE OF 1 TO 61.

22 = DATUM INVALID FOR TYPE 3 KEYWORD.

SPECIAL CONSIDERATIONS AND ASSUMPTIONS:
1. INTEGERS OCCUPY ONE 24 BIT WORD, REALS TWO WORDS, AND 
   FOUR 6-BIT CHARACTERS OCCUPY ONE WORD.
2. THE FIRST CHARACTER IS LEFT JUSTIFIED. THIS FACT IS USED 
   IN THE 4TH WORD OF AN ISA ARRAY ENTRY.
3. TWO'S COMPLEMENT ARITHMETIC, WITH ZERO AND MINUS 
   ZEROS DEFINED BUT FUNCTIONALLY EQUAL.
4. ONLY PARAMETER ADDRESSES ARE PASSED IN LINKAGE. IN THE CASE 
   OF REAL PARAMETERS, THE ADDRESS OF THE FIRST OF THE TWO WORDS 
   OF THE REAL NUMBER IS PASSED. THUS A PASSED PARAMETER MAY 
   CHANGE TYPE IN USAGE ONLY FOR THE 
   CONTENTS REMAIN THE SAME AND MUST HANDLED PROPERLY.
5. DIMENSION OF CPBUFF IN RECGET SUBROUTINE MUST BE .GE. 
   MAXLEN*ISHIFT. (ISHIFT MAY BE NEGATIVE).

SUBROUTINE VARDEF(ISA,VSYM,VADDR,JVTYPE,JV01,JV02,JV03,NUM)

THIS ROUTINE BUILDS THE VARIABLE DEFINITION ARRAY.

THE FIRST WORD OF THE ARRAY IS A POINTER TO THE 
WORD TO STORE THE NEXT DEFINITION. A DEFINITION 
CONSISTS OF AN EIGHT WORD GROUP. THE FIRST TWO 
WORDS OF THE GROUP CONTAIN THE CHARACTERS TO BE 
SEARCHED FOR (LEFT JUSTIFIED WITH BLANK FILL, 
ONLY ALPHA CHARACTERS AND DIGITS ARE ALLOWED AND 
IT MUST START WITH AN ALPHA CHARACTER). THESE 8 CHAR-
ACTERS ARE REFERRED TO AS THE KEYWORD. THE 
3RD WORD IN THE ARRAY HOLDS THE ADDRESS OF THE 
CALLING PROGRAM VARIABLE ASSOCIATED WITH THE KEYWORD. 
THE 1ST CHARACTER OF THE FOURTH WORD CONTAINS A CODE 
DESCRIBING THE TYPE OF VALUE TO BE STORED IN 
THE VARIABLE AS FOLLOWS.

1 = INTEGER VARIABLE OR ARRAY 
2 = REAL (TWO WORD) VARIABLE OR ARRAY 
3 = CHARACTER VARIABLE OR ARRAY 
4 = INTEGER (ONE WORD) SYMBOL 
5 = REAL (TWO WORD) SYMBOL 
6 = REAL (TWO WORD) SPECIAL SYMBOL - ANYTHING 
   VALID, ENDED BY A SPACE, MUST BE 8 CHARA-
7 = NO VALUE STORED - ONLY PRESENCE OF SYMBOL INDICATED. (SEE BELOW)

8 = VARIABLE LENGTH ITEM. FIRST DIMENSION INDICATES MAXIMUM NUMBER OF CHARACTERS TO BE STORED. IF MORE ARE SPECIFIED ON THE INPUT RECORD, THEY ARE IGNORED. IF FEWER ARE SPECIFIED, THE VARIABLE IS BLANK FILLED UP TO ITS MAXIMUM.

ARRAY FORMS -
A(1) = 12.43    12.4E-6
B(3) = 2.4
C(5) = 2.4
 FOR B DIMENSIONED AS 3 AND C DIMENSIONED AS 5 AND JJ DIMENSIONED AS 3,4.
JJ(1) = (12 NUMBERS)
JJ(2) = (THREE NUMBERS FOR THE 2ND ROW)
JJ(3,3) = 45

9 = LOGICAL UNIT NUMBER. VALUE ASSIGNED TO VARIABLE MAY BE AN INTEGER OR A FILE NAME. IF A FILE NAME, AN UNUSED LUN IS EQUIPPED, IF POSSIBLE, TO THE FILE AND THE NUMBER OF THE LUN IS STORED. IF THE FILE NAME WAS PREVIOUSLY SPECIFIED (THE LAST THREE CHARACTERS OF THE FOURTH WORD IN THE ARRAY IS THE INDICATOR), IT HAS A VALUE OF 1. THE LUN PREVIOUSLY EQUIPPED IS FIRST UNEQUIPPED.


IF THE DIMENSION SPECIFICATIONS IN WORDS 5 THRU 7 ARE "GT. 5000 (ABS VALUE), THE PROCESSING ROUTINE ASSUMES THAT THE SPECIFICATION IS DEFINED BY A VARIABLE ASSOCIATED WITH ANOTHER KEYWORD IN THE TABLE. IF FOUND AND IF THAT KEYWORD WAS ALREADY ACCESSED, THE VALUE STORED FOR THAT KEYWORD WILL BE USED AS A SPECIFICATION OF DIMENSION.

ZERO IS RETURNED IN NUM IF ERRORS 3 OR 4 HAS OCCURRED.

THE FIFTH, SIXTH, AND SEVENTH WORDS OF THE GROUP SPECIFY THE DIMENSION OF THE VARIABLE (LEGAL ON TYPES 1, 2, 3, 8 ONLY). ZERO INDICATES NO DIMENSION, OTHERWISE THE VALUE IS THE MAXIMUM (SIMILAR TO A DIMENSION STATEMENT).

THE LAST 3 CHARACTERS OF THE FOURTH WORD IS A COUNTER INDICATING HOW MANY TIMES A
VARIABLE WAS REFERENCED. SEE ABOVE FOR EXCEPTION FOR TYPE 9 VARIABLES.
The last word of the array holds a user assigned integer.
ISA = ARRAY FOR DEFINITIONS
X = VSYM = a character identification symbol, called the keyword, to be searched for.
VADDR = ADDRESS WHERE DATA IS TO BE STORED (FIRST ADDRESS FOR ARRAYS)
JVTYPE = IVTYPE = VARIABLE TYPE CODE
IVD1 = JVOD1 = , etc., = ARRAY DIMENSIONS
NUM = USER NUMBER ASSOCIATED WITH SYMBOL. THIS NUMBER IS RETURNED TO THE USER BY INPROC whenever
the keyword in VSYM is found (the keyword itself
is also available to the user from INPROC)

DIMENSION ISA(750), IX(2), CISA(4)
CHARACTER CISA
EQUIVALENCE (X, IX), (ISAC, CISA)

CONTINUE
IVTYPE=JV TYPE
IVD1=JVOD1
IVD2=JVOD2
IVD3=JVOD3
LDUP=1
X=VSYM

IF DEFINITION ARRAY NOT YET ACCESSED, INITIALIZE
ARRAY POINTER TO POINT TO FIRST OPEN ENTRY
IF (ISA.LE.1) ISA=2
SEARCH ARRAY FOR DUPLICATE KEYWORD. IF FOUND, J
POINTS TO DUPLICATE ENTRY
CALL VARFIND (ISA, VSYM, J)
JUMP IF DUPLICATE
IF (J.NE.0) GO TO 30
MARK AS NOT DUPLICATE KEYWORD, AND SET J=POINTER.
J=ISA
LDUP=0
JUMP ACCORDING TO VARIABLE TYPE
GO TO (60, 40, 40, 40, 40, 40, 40, 40, 40, 110) TABS(IVTYPE)-2
TYPES 4, 5, 6, 7. JUMP TO ERROR ROUTINE IF DIMEN-
SIONED
40 IF (IVD1.NE.0) 100, 60
JUMP TO ERROR IF OTHER THAN 1 DIMENSION GIVE
TYPE 8. JUMP TO ERROR IF OTHER THAN 1 DIMENSION GIVE
100 STORE INFORMATION IN DEFINITION ARRAY
60 ISA(J)=IX(1)
EXAMINE DIMENSIONS. JUMP TO ERROR IF LOWER DIMENSION IS ZERO WHEN HIGHER DIMENSION IS NOT.

IF (IV03 .EQ. 0) GO TO 71

IF (IV02 .GT. 0) 100,90

70 IF (IV02 .NE. 0 .AND. IV01 .LE. 0) GO TO 100

C STORE DIMENSIONS IN DEFINITION ARRAY

90 ISA(J+4) = IV01
   ISA(J+5) = IV02
   ISA(J+6) = IV03

C IF NOT DUPLICATE KEYWORD, BUMP AND STORE ARRAY POINTER.

IF (LDUP .GT. 0) ISA = J+1
RETURN

IF (LDUP .EQ. 0) ISA = J+1
RETURN

END

SUBROUTINE KVARDEF(ISAY,SYM)

THIS ROUTINE REMOVES AN ENTRY, IF PRESENT, FROM THE DEFINITION ARRAY, AND COMPRESSES THE ARRAY.

DIMENSION ISA(750), IX(2)

EQUIVALENCE (TX,X)

X = SYM

SEARCH FOR KEYWORD, RETURN IF NOT FOUND. J POINTS TO ENTRY.

CALL VARFIND(ISAY,X,JJ)

MOVE REMAINING ENTRIES IN ARRAY DOWN ONE SLOT (8 WORDS) TO FILL HOLE.

IF (JJ .EQ. 0) RETURN

IFEND = ISA(1)
JEND = IFEND - B

20 ISA(K) = ISA(K+8)
BLANK NOW UNUSED SLOT AT END OF ARRAY

DO 30 K=IEND,JEND
      ISA(K)=4H
      RETURN
END

SUBROUTINE VARFIN(ISA,SYM,J)
      DIMENSION ISA(750),IX(3)
      EQUIVALENCE (IX,X)
      X=SYM
      JEND=ISA(1)-1
      SEARCH FOR KEYWORD.
      J=0
      DO 10 I=2,TEND,8
      IF (IX(1).NE.ISA(I)) GO TO 10
      IF (IX(2).NE.ISA(I+1)) GO TO 10
      J=I
      RETURN
10  CONTINUE
      END

SUBROUTINE INPROC(ISA,CBUFF,JLUN,JEFO,JLUN,MAXLEN,SYM,NJM,JERR,*
                    JOPT,MARK)
      THIS ROUTINE PROCESSES THE SYMBOL IN CBUFF POINTED TO BY IC. IF
      THE SYMBOL IS UNDEFINED, OR ERRORS ARE FOUND WHILE
      PROCESSING IT, THE ROUTINE WILL PERFORM ACCORDING TO
      THE CODE IN JOPT AS FOLLOWS -
1 = PRINT ERROR AND ABORT
2 = PRINT ERROR MESSAGE AND RETURN
3 = RETURN WITHOUT PRINTING ERRORS
4 = PRINT ALL ERRORS EXCEPT UNDEFINED SYMBOL
5 = SAME AS 2 BUT ALLOWS FOR TTY RECOVERY IF INPUT IS
   FROM TTY
6 = SAME AS 4 BUT ALLOWS FOR TTY RECOVERY IF INPUT IS
   FROM TTY
7,8 = SAME AS 5,6 EXCEPT IF INPUT FROM TTY, ROUTINE
     WILL PRINT ONLY A SHORT MESSAGE ON INPUT ERRORS.
   ENTRY OF A DOUBLE DOLLAR SIGN, $A, WILL CAUSE
THE ROUTINE WILL START PROCESSING THE RECORD IN CBUFF AT THE
POSITION POINTED TO BY IC UNLESS IC IS GT. MAXLEN. IF
IC IS GT. MAXLEN THE ROUTINE WILL READ A NEW RECORD
AND
START PROCESSING IN COLUMN 1 OF THE NEW RECORD. IF
THE USER WISHES TO SUPPLY THE INFORMATION TO BE PRO-
CESSSED TO THE ROUTINE RATHER THAN HAVE THE ROUTINE
READ THE RECORD, HE MUST SPECIFY A LUN GT. 100, AND
PLACE THE INFORMATION TO BE PROCESSED IN CBUFF. WHEN
MORE INFORMATION IS REQUIRED (THE ROUTINE HAS
PROCESSED LAST (MAXLEN) CHARACTER IN CBUFF) THE ROUTINE
WILL RETURN WITH IFOF = -1. THE USER MUST CHECK IERR TO
SEE IF AN ERROR HAS OCCURRED. IF NO ERROR, OR AFTER
THE ERROR IS ACCOUNTED FOR (OR IGNORED) BY THE USER,
HE MUST THEN REFILL CBUFF AND CALL THE SUBROUTINE
WITHOUT ALTERING IFOF. IF HE WISHES THE ROUTINE TO
CONTINUE, INPUT MAY BE TERMINATED WITH AN EOF SYMBOL
(DOUBLE LEFT BRACKET - OCTAL 1717) PLACED IN THE INPUT
BUFFER.

IF IERR IS NEGATIVE ON ENTRY, THE
ROUTINE WILL SEARCH FOR THE FIRST KEYWORD
THE USER MUST ALLOW FOR CBUFF TO BE DIMENSIONED AS A
CHARACTER ARRAY OF LENGTH CBUFFRLEN.

A SPACE IS VALID AFTER A KEYWORD, THE FOLLOWING
DELIMITERS ( ) , = AND AFTER A NUMBER. SPACES MAY
NOT BE IMBEDDED IN NUMBERS OR KEYWORDS. A KEYWORD,
DEFINED BY VARDEFI CONSISTS OF ONLY ALPHANUMERIC
CHARACTERS AND MUST START WITH AN ALPHA,
A BLANK OR A SECOND + OR A SECOND * UNLESS
IT FOLLOWS THE EXPONENT INDICATOR F, OR AN ALPHA
CHARACTER OTHER THAN E, OR THE SECOND E,
TERMINATES A NUMBER.

SPECIAL INPUT SYMBOLS -
: (END OF FILE) - NO MORE INPUT.
? (QUESTION MARKS) ROUTINE WILL PRINT EXPANDED ERROR
MESSAGE FOR LAST ERROR.
> (Tells ROUTINE TO reset and expect
A SYMBOL TO FOLLOW.
ISA = ARRAY OF KEYWORD DEFINITIONS
CBUFF = INPUT BUFFER
JLUN = LUN = INPUT LUN
JEOF = IFOF = END OF FILE FLAG
JC = IC = CHARACTER INDEX OF CBUFF
MAXLEN = INPUT BUFFER LENGTH (MAX OF 136 FOR CDC 3300)
XSYM = VARIABLE HOLDING KEYWORD PROCESSED BY INPROC
ON RETURN.
NUM = NUMBER ASSOCIATED BY USER WITH SYMBOL IN VARDEF ROUTINE.
JERR = IERR = ERROR CODE
IOPT = ILOPT = USER OPTION
MARK = USER INTERACTIVE PROMPT CHARACTER OR MESSAGE CODE. IF LT. 0, NO PROMPT CHARACTER IS PRINTED, IF .LE. 0, THE STANDARD PROMPT CHARACTER, >, IS PRINTED. IF .GT. 0, THE CONTENTS OF CBUFF FROM POSITION 1 TO MARK ARE PRINTED JUST PRIOR TO INPUT FROM A USER INTERACTIVE DEVICE (CRT, TTY, ETC.). IF CBUFF(MARK+1) AND CBUFF(MARK+2) CONTAIN THE CODES $" (DOUBLE DOLLAR SIGN) A CARRIAGE RETURN AND LINE FEED IS SENT TO THE IO DEVICE.

 Character C, CC, CBUFF, CISA
 Integer HARDWARE
 DIMENSION ISA(750), CBUFF(140), IX(2), IY(4), JD(3), SYM(3)
 DIMENSION CI(16), XI(2), JDY(3), IBUFF(12), CISA(4)
 DIMENSION ABORTMSG(4), IRMSG(3), RVMSG(12)
 EQUIVALENCE (X, IX), (Y, IY), (C, XI), (CISA, ISAC)
 DATA ((ABORTMSG(I), I=1, 4) = 8M0ABN0PMA, 8HL TERMIN, 8HATION OF * , 8H PROGRAM)
 DATA ((IRMSG(I), I=1, 3) = 4H - REENT, 4H4ER )
 DATA ((RVMSG(I), I=1, 2) = 8H - REENT, 8H4ER VALUE)
 DATA (BASESYM=5H )
 DATA (SYM=5H )
 DATA (PSYM=5H )
 DATA (LICSTRT=0 )
 ASSIGN GLOBAL VARIABLES TO LOCAL VARIABLES
 CONTINUE
 RETURN IF END-OF-FILE ON INPUT
 IF (IEOF, LT. 0) GO TO 150
 LUN=JLUN
 IEOF=IEOF
 IC=JC
 IERR=JERR
 IOPT=ILOPT
 IRSCLEN=MAXLEN
 DFLCHAR=4H=
 IEOF=0
 IF ROUTINE NOT REQUESTED TO FIND FIRST VALID KEYWORD, (IE., NEGATIVE) RESET ERROR CODE.
 IF (IERR, GE. 0) IERR=1
RESET REENTRY CODE.

SET CODE TO RETURN TO THIS PART, KEYWORD NUMBER TO DEFAULT, AND JUMP TO ROUTINE TO GET FIRST SYMBOL FROM INPUT FILE.

IRTN=1
NUM=0
GO TO 450

SAVE ITEM FOUND, AND SAVE POINTER TO DEFINITION ARRAY ENTRY FOR PREVIOUS KEYWORD (JJY).

ITNY=1
BASESYM=SYM
JJX=JJ

RETURN IF END-OF-FILE ON INPUT.

IF (EOF,GT,0) GO TO 160
IF COMMA FOUND, GET NEXT ITEM.

IF (SYM.EQ.5H,) GO TO 5
IF ITEM NOT AN ALPHANUMERIC SYMBOL, JUMP TO ERROR SEARCH DEFINITION ARRAY FOR MATCH WITH KEYWORD.

CALL VARPIND(ISA,SYM,JJ)
JUMP IF NOT FOUND - SYMBOL NOT A KEYWORD.

IF (JJ.EQ.0) GO TO 510
GET NUMBER OF KEYWORD, SAVE KEYWORD, AND RESET ERROR CODE.

NUM=ISA(JJ+7)
BASESYM=SYM
IERR=1

GET DIMENSIONS OF VARIABLE ASSOCIATED WITH KEYWORD.

DO 390 KK=1,3
JK=JJ+KK
JDX(KK)=ISA(JK+3)
390
IF DIMENSION SPECIFIED BY A SYMBOL, GO TO STATEMENT 380

IF (JDX(KK).GT.50000.0J0X(KK).LT.0) 380,390
SEARCH DEFINITION ARRAY ON SYMBOL

IX(2)=4H
IX=JDXX(KK)
CALL VARPIND(ISA,X,L)
JUMP TO ERROR IF SYMBOL NOT A KEYWORD.

IF (L.EQ.0) GO TO 660
IF NO DATA STORED FOR SPECIFICATION VARIABLE.

CALL ROUTINE TO GET VALUE AND STORE IT AS DIMENSION FOR VARIABLE CURRENTLY BEING PROCESSED.

JDXX(KK)=IVARGET(ISA(L+7))
MAKE DUPLICATE ARRAY OF DIMENSIONS.
C  GET VARIABLE TYPE.
ISAC=ISA(JJ+3)

MYPF=ISA(1)

C  JUMP IF ACCESS TO VARIABLE IS NOT RESTRICTED TO ONE TIME ONLY.
IF (MYPF.GT.0) GO TO 11

C  SET VARIABLE TYPE CODE TO POSITIVE VALUE.
MYPF=-MYPF

C  JUMP IF LAST VARIABLE PROCESSED SAME AS CURRENT.
IF (MYPF.GT.0) GO TO 11

C  JUMP IF VARIABLE ALREADY ACCESSED.
IF (AND(ISA(JJ+4),SYM.EQ.'SYNLMST') .LT. 0) GO TO 11

C  SET USER ENTERED DIMENSIONS ARRAY TO 0.
J0(1)=J0(2)=J0(3)=0

C  JUMP ACCORDING TO VARIABLE TYPE.
GO TO (12,40,40,40,150,12,40,150) MYPF-2

C  TYPES 1,2,3 AND 8, JUMP IF VARIABLE UNDIMENSIONED.


C  THE VALUES OF I,J, AND J WHEN PRESENT, ARE STORED IN ARRAY JO.

C  GET LEFT PARENT.
IRTN=3

C  JUMP IF NOT PROCESSING A TYPE 8 VARIABLE - OF THE FORM KEYWORD, I,= OR KEYWORD=.
GO TO 450

C  IF (MYPF.NE.8) GO TO 14
IF (SYM.EQ.'5H,) GO TO 15
IF (SYM.EQ.'DECHAR) GO TO 205

C  JUMP TO ERROR IF LEFT PARENTHESES NOT FOUND.

C  JUMP IF TOO MANY SUBSCRIPTS.
DO 20 KK=1,3
IF (SYM.EQ.'DECHAR) GO TO 1501
IF (SYM.EQ.'TH( )') GO TO 620

C  GET UP TO 3 SUBSCRIPTS.

C  JUMP IF SUBSCRIPT OMITTED FROM INPUT STREAM.
IRTN=4
GO TO 450

C  NO PARENS - NO ARRAY SPECS.
IF (SYM.EQ.'5H,) GO TO 520
GO TO 456

C  JUMP IF SUBSCRIPT OMITTED FROM INPUT STREAM.
JUMP IF NOT END OF EXPRESSION.

IF (SYM .NE. 5)
    GO TO 16

JUMP TO ERROR ROUTINE IF TOO FEW SUBSCRIPTS SPECIFIED

IF (JDX(KK+1) .NE. 0) 560, 30

JUMP IF SUBSCRIPT NOT NUMERIC.

CALL ROUTINE TO RIGHT ADJUST NUMBER, THEN CONVERT IT TO BINARY INTEGER.

CALL CONVERT(SYM, 1, IDX)

IF (IDX) 17, 540, 18

JUMP TO ERROR IF NUMBER EXCEEDS DIMENSION.

IF (IDX .GT. JOX(KK)) GO TO 550

STORE NUMBER IN SUBSCRIPT ARRAY.

JO(KK) = IOX

GET SEPARATING COMMA

IPTN = 5

GO TO 450

455 IF (SYM .EQ. 5M,
    ) GO TO 19

TEST FOR END OF EXPRESSION.

IF (SYM .EQ. 5H,
    ) 30, 520

JUMP IF 1ST SUBSCRIPT PROCESSED ON TYPE 8 VARIABLE.

IF (KX .EQ. 1.AND. MTYPE .EQ. 6) GO TO 120

CONTINUE

GET DELIMITING CHARACTER

CONTINUE

IRTN = 6

GO TO 451

IF (SYM .NE. DELCHAR) GO TO 650

THE FOLLOWING SECTION OF CODE SETS UP THE TRIPLE DO LOOP TO READ IN DATA AS A TRIPLE DIMENSIONAL ARRAY.

AN UNDIMENSIONED VARIABLE IS ASSUMED TO BE DIMENSIONED 1, 1, 1, ETC.

IF A SUBSCRIPT WAS NOT SPECIFIED IN THE INPUT STRING, SAY THE LTH, (IE. JD(L) = 0) THEN THE CORRESPONDING DO LOOP IS SET UP TO RUN FROM 1 TO THE SUBSCRIPT=S MAXIMUM VALUE (FROM JDX(L)) - THE VALUE FOR THE LTH DIMENSION FROM DEFINITION ARRAY. OTHERWISE THE DO LOOP IS SET UP TO RUN FOR ONE ITERATION ONLY, FROM THE SUBSCRIPT SPECIFIED TO ITSELF.

EXAMPLES - FOR (, 1, 10) WHERE THE MAXIMUM VALUE FOR THE 1ST DIMENSION IS 5, THE FOLLOWING DO LOOPS ARE GENERATED...
I=1,5
J=10,10
K=1,1

FOR AN UNSUBSCRIPTED VARIABLE

I=1,1
J=1,1
K=1,1

M1=M2=M3=1
N1=JDX(1)
N2=JDX(2)
N3=JDX(3)

IF (JDX(1).NE.0) M1=JDX(1)
IF (JDX(2).NE.0) N2=JDX(2)
IF (JDX(3).NE.0) N3=JDX(3)

IF (N1.EQ.0) N1=1
IF (N2.EQ.0) N2=1
IF (N3.EQ.0) N3=1

COMPUTE ARRAY POINTER BASE.

IDX=I+JDX(1)*JDX(1)*JDX(2)

DO 110 I=M1,N1
DO 110 J=M2,N2
DO 110 K=M3,N3

RUMP COUNTER WORD IN DEFINITION ARRAY EXCEPT FOR VARIABLE TYPE 9.

IF (MTYPE.NE.9) ISA(JJ+3)=OR(AND(ISA(JJ+3),4H000000)) * 1,AND(ISA(JJ+3),4H000000))

COMPUTE POINTER TO STORE INFORMATION IN STANDARD FORTRAN FORM, BY COLUMNS.

IDX=I+JDX(1)*JDX(1)*JDX(2)*K-IDX3

GET VALUES

IF TYPE 3 VARIABLE (CHARACTER), SET TO GET ONLY ONE CHARACTER FROM INPUT.

43 IRTN=2
GO TO 450

452 ITTY=2
GO TO (50,30,90,100,510,510,510,50) MTYPE-1, TYPE 1, 2, AND 9, NUMERIC OR LUN, SKIP INTERVANING COMMAS.

IF (SYM.EQ.5H) GO TO 43
JUMP IF REAL DATA CALLED FOR.

IF (MTYPE.EQ.2) GO TO 50
JUMP IF INTEGER ONLY CALLED FOR.

IF (MTYPE.NE.9) GO TO 51
JUMP IF VARIABLE DOESN'T HOLD LUN EQUIPPED BY THIS ROUTINE.
IF (AND(ISA(JJ+3),4H000000).EQ.0) GO TO 51
RESET EQUIPPED INDICATOR AND UNEQUIP THE LUN.

ISA(JJ+3)=AND(ISA(JJ+3),4H000000)
CALL UNEQUP(TARGET(ISA(JJ+2)))

GO TO (52,55,570) ICOOE
SYMBOL IS ASSUMED TO BE FILE NAME.

IF (MTYPE.NE.9) GO TO 570
JUMP IF THE DATUM IS SYMBOL, INTEGER OR OTHER.

IY=0
GET AND EQUIP AN UNUSED LUN TO THE FILE NAME.

CALL SAFOD(IY,SYM)
GET AND EQUIP AN UNUSED LUN TO THE FILE NAME.

IF (IY.LE.0.OR.IY.GE.63) GO TO 580
SET INDICATOR TO SHOW UNIT WAS EQUIPPED BY THIS
ROUTINE.

ISA(JJ+3)=AND(ISA(JJ+3),4H000000)+1
GO TO 57
CALL ROUTINE TO RIGHT JUSTIFY INTEGER AND THEN CON-
COMPUTE STORAGE INDEX AND STORE DATUM IN VARIABLE.

CALL VARSTOR(IADDRY, Y, 4)
GO TO 110

CHARACTER DATUM. COMPUTE STORAGE INDEX, GET CHARACTER
AND STORE IN VARIABLE.

IADDRY = (IODX/4) + ISA(JJ+2)
IF (ICODE.GE.3) GO TO 620

XC = SYM
IY = C(1)
IF (C(2).EQ.1R) GO TO 66
IF (C(1).GE.10) GO TO 620
IF (C(3).NE.1R) GO TO 620
CALL CONVERT(SYM, 1, IY)
IF (IY.GE.65) GO TO 620
CALL VARSTOR(IADDRY, IY, 0)
GO TO 110

FOUR CHARACTER SYMBOL, TYPE 4 VARIABLE.
JUMP IF ITEM FETCHED NOT A SYMBOL.

IF (ICODE.NE.1) GO TO 600
CALL VARSTOR (ISA(JJ+2), SYM, 2)
GO TO 160

FIGHT CHARACTER SYMBOL - TYPE 5 VARIABLE. JUMP TO
ERROR IF NOT SYMBOL.

IF (ICODE.NE.1) GO TO 600
CALL VARSTOR (ISA(JJ+2), SYM, 4)
GO TO 160
CONTINUE
GO TO 160

TYPE 6, SPECIAL CHARACTER, MISCELLANEOUS. LEFT JUSTI-
FY CHARACTER IMMEDIATELY FOLLOWING DELIMITING EQUAL
SIGN IN INPUT RECORD INTO ARRAY XC. STOP WHEN BLANK
ENCOUNTERED.

XC = 8H
GO 130 I=1, 9
IF (CBUFF(IC).EQ.1R) GO TO 140
C(I) = CBUFF(IC)
IC = IC+1
CONTINUE
ERROR - MORE THAN 8 CHARACTERS.
GO TO 640
STORE CHARACTERS.
CALL VARSTOR (ISA(JJ+2), XC, 4)
TYPE 7 VARIABLE. RUMP KEYWORD COUNTER.
CISA(JJ+3)=OR(AND(IISA(JJ+3), 4H0000))+1, AND(ISAI(JJ+3), 4H0000))

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C BSX=LAST=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C IRTN=0

ASSIGN LOCAL VARIABLE TO GLOBAL VARIABLE AND RETURN.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C IC=IC

ASSIGN LOCAL VARIABLE TO GLOBAL VARIABLE AND RETURN.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.

C JEOF=IEOF

SAVE KEYWORD CURRENTLY BEING PROCESSED.

C XSYM=BASFSYM

RESET RETURN FROM ITEMGET SWITCH.
C(IST)=CC
IST=IST+1
JUMP IF WORD NOT FULL.

IF (IST.LT.5) GO TO 230
STORE WORD, BUMP VARIABLE ARRAY INDEX, RESET WORD
WORD CHARACTER INDEX.

CALL VARSTOR(IADDRY,XC,2)
IADDRY=IADDRY+1
XC=RH
IST=1
IF WORD PARTLY FILLED, STORE IT.
CONTINUE

IF (IST.NE.1) CALL VARSTOR(IADDRY,XC,2)
GO TO 150
CALL ROUTINE TO FETCH NEXT ITEM FROM INPUT STRING.

CALL ITEMGET(CBUFF,LUN,IEOF,IC,MAXLEN,ICSTRT,SYM,ICODE,ISRCLEN,
* MARK)
IF (IEOF.LT.0) RETURN
IF (SYM.EQ.5H) GO TO 100
IF (SYM.EQ.5H) GO TO 460
IF (SYM.EQ.5H) GO TO 459
GO TO (451,452,453,454,455,456,457) IRTN
PRINT ADDITIONAL ERROR MESSAGE.
CALL PGEROR(LIERR,PSYM,PSYM,ICSTRR)
SKIP IF OP A

GO TO 160
SYMBOL NOT FOUND

IF (IFRR.LT.0) GO TO 7
IFRR=4
GO TO 700
SYMBOL NOT DEFINED

IF (IFRR.LT.0) GO TO 5
IFRR=4
GO TO 700

VALID CHARACTER

IF (IFRR.LT.0) GO TO 5
IFRR=5
GO TO 700

GO TO 700
NON INTEGER SUBSCRIPT

GO TO 700
SUBSCRIPT .GT. 32767 OR .LE. 0
GO TO 700
SUBSCRIPT EXCEEDS SPECIFIED RANGE

INVALID DIMENSION

INVALID VALUE - MUST BE INTEGER

INVALID VALUE - MUST BE REAL

CONTINUE

IF (ICODE.NF.1) GO TO 700

SEARCH TO SEE IF ITEM FOUND IS A KEYWORD.

CALL VARFIND(ISA,SYM,JJ)

ERROR - NOT A KEYWORD.

IF (JJ.EQ.0) GO TO 700

RESET IC SO THAT NEXT ENTRY TO THIS ROUTINE WILL

START PROCESSING THIS NEW KEYWORD.

IC=ICSTRT

RESET ERROR CODE AND RETURN.

INVALID VALUE - MUST BE SYMBOL

SYMBOL DISALLOWED

INVALID CHARACTER DATUM.

SYMBOL TOO LONG FOR TYPE MISC

NO = SIGN FOUND

INVALID DEFINITION OF ARRAY BOUNDS

ANOTHER VARIABLE MUST FIRST BE SPECIFIED

UNABLE TO EQUIP SAVED FILE
01151 C    GO TO 590  
01152 C    SAVE ERROR CODE FOR USE IF ^A ENTERED.
01153 700 LERR=IERR
01154 PBSYM=BASESYM
01155 PSYM=SYM
01156 LICSTRT=ICSTRT
01157 C    JUMP ACCORDING TO USER'S OPTION ON ERROR.
01158 C    GO TO (720,160,710,715,710,715,710) IOPT-1
01159 C    JUMP TO RETURN IF ERROR WAS UNDEFINED SYMBOL.
01160 710 IF (IERR.EQ.3) GO TO 160
01161 715 IF (IOPT.GE.7) GO TO 730
01162 C    CALL ROUTINE TO PRINT ERROR MESSAGE.
01163 720 CALL PGERROR(IERR,BASESYM,SYM,ICSTRT)
01164 C    IF (IOPT.GE.5) GO TO 730
01165 C    IF (IOPT.NE.1) GO TO 160
01166 C    CALL FWRITE(6111,ABORTMSG)
01167 C    RETURN IF NOT TTY OR CRT.
01168 730 IF (HARDWARE(LUN).EQ.6.OR.HARDWARE(LUN).EQ.9) 740,735
01169 735 IF (IOPT.GE.7) CALL PGERROR(IERR,BASESYM,SYM,ICSTRT)
01170 740 IC=140
01171 C    JUMP IF ERROR NOT OF DATUM.
01172 C    IF (IERR.EQ.10) GO TO 750
01173 C    IF (IERR.EQ.19) GO TO 750
01174 C    IF (IERR.EQ.12) GO TO 750
01175 C    IF (IERR.EQ.9) GO TO 750
01176 C    IF (IERR.EQ.22) GO TO 750
01177 C    CALL EWRITE(61,3,IPMSG)
01178 C    PRINT MESSAGE AND GET NEW DATUM.
01179 C    RESF ERROR INDICATOR (IN HOPES NEXT ENTRY WILL BE
to correct error)
01180 C    IERR=1
01181 JC=MAXLEN+1
01182 5 GO TO 5
01183 C    PRINT MESSAGE AND GET NEW DATUM.
01184 750 CALL FWRITE(61,4,RVMSC)
01185 C    RESF ERROR INDICATOR
01186 C    IERR=1
01187 JC=MAXLEN+1
01188 43 GO TO 43
01189 C    IF (IERR+LT.0) RETURN
01190 800 IEOF=1
01191 SYM=SYM
01192 ICONE=1
01193 IF (IERR.LT.0) RETURN
THIS ROUTINE FETCHES THE NEXT ITEM FROM THE BUFFER, STARTING WITH THE CHARACTER POINTED TO BY IC AND RETURNS WITH A TYPE CODE IN ICODE. LEADING BLANKS ONLY ARE IGNORED. THE CODES FOR ICODE ARE:

1 = SYMBOL
2 = INTEGER
3 = REAL
4 = SPEC CHAR
5 = ERROR / NOTHING FOUND

IEOF = -1 INDICATES END OF RECORD - USER MUST REFILL BUFFER.

ISA = SYMBOL DEFINITION ARRAY
CBUFF = INPUT BUFFER
LUN = INPUT UNIT
IEOF = EOF INDICATOR OR END OF BUFFER INDICATOR
IC = CBUFF INDEX POINTER
SYM = ARRAY HOLDING ITEM Fetched
ISCHLEN = LENGTH BUFFER IS TO BE SEARCHED

SUBROUTINE ITFGET(CBUFF,LUN,IEOF,IC,MAYLEN,ICSTRT,SYM,ICODE, ISCHLEN,MARK)

CHARACTER CBUFF,CSYM
DIMENSION CBUFF(140),CSYM(24),RSYM(3),SYM(3)

EQUIVALENCE(RSYM,CSYM)
CONTINUE
IF EXIT FROM THIS ROUTINE WAS TO GET MORE INPUT (IEOF=-1), RETURN TO POINT OF EXIT.

IF (IEOF.LT.0) GO TO (10,90) IRTN

IF (IEOF.LT.0) GO TO (10,90) IRTN

IREN=0
IST=1
RSYM=SYM(2)=SYM(3)=8H
ICODE=5
IEOS=IEOF=0
IC=IC+1
ICEND=IC+ISCHLEN
IF (ICEND.LT.IC) GO TO 200

CALL EBUMP(CBUFF,LUN,IEOF,IC,MAYLEN,MARK)
ICSTRT=IC

IF (IEOF) 15,20,190
IRTN=1
RETURN
JUMP ACCORDING TO TYPE OF FIRST CHARACTER FETCHED, AND SET CODE INDICATING TYPE OF ITEM (SYMBOL, NUMBER, ETC.)

GO TO (30, 40, 50, 40, 10, 60) ITYPE (CBUFF (IC))

ALPHA CHARACTER

GO TO 70

NUMERIC DIGIT

GO TO 70

PLUS OR MINUS

GO TO 70

DECIMAL POINT

GO TO 70

SPECIAL CHARACTER

STORE FETCHED CHARACTER.

CSYM (IST) = CBUFF (IC)

IF (ICODE .NE. 1) GO TO 80

JUMP IF OTHER THAN A SYMBOL BEING PROCESSED.

IF (ICODE .NE. 1) GO TO 70

IF SYMBOL AT 8 CHARACTERS, DECREMENT INDEX SO A

9TH (OR MORE) CHARACTER WILL BECOME THE LAST CHAR-

ACTER - THE SYMBOL ABC0567X IS THE SAME AS A3C0567X

IF (IST .GE. 1) IST = 7

BUMP STORAGE INDEX.

IST = IST + 1

GET NEXT CHARACTER

IF (ICODE .LE. IC) GO TO 200

CALL FBUMP (CBUFF, LUN, IEOF, IC, MAXLEN, MARK)

TEST FOR MORE INPUT NEEDED (NEGATIVE VALUE)

IF (IEOF) 95, 97, 190

RETURN

JUMP ACCORDING TO TYPE OF CHARACTER FETCHED.

GO TO (100, 110, 120, 130, 200) ITYPE (CBUFF (IC))

ALPHA. JUMP TO GET NEXT CHARACTER IF PROCESSING

SYMBOL.

IF (ICODE .EQ. 1) GO TO 70

TEST FOR EXPONENT LETTER E IF PROCESSING A REAL

NUMBER.

IF (ICODE .NE. 3) GO TO 200

IF (CBUFF (IC) .NE. 'E') GO TO 200

JUMP OF E ALREADY FOUND.

IF (ICODE .EQ. 0) GO TO 200

SET E INDICATOR SWITCH ON.
SUBROUTINE FIBUMP(CBUFF,LUN,IEOF,IC,MAXLEN,MARK)

   This routine bumps the fetch index, IC, and checks for end of search and end of record.
   If user supplying input, set IEOF to -1 to indicate end of record and return to users routine
   CBUFF = input buffer
   LUN = input unit
   IEOF = eof indicator or end of buffer indicator
   IC = CBUFF index pointer
   IEOFS = end of search indicator
   ISPCHLEN = length buffer is to be searched

DIMENSION CBUFF(140)
CHARACTER CBUFF
CONTINUE
JUMP IF NOT END OF FILE ON INPUT.
IF (IEOF.GE.0) GO TO 10
IEOF=0
RETURN
JUMP IF NOT END OF INPUT RECORD.
IC=IC+1
JUMP IF NOT USER SUPPLIED INPUT.
IF (LUN.LE.10) GO TO 20
SET CODE TO REQUEST USER TO SUPPLY MORE INPUT.
IEOF=-1
RETURN
CALL ROUTINE TO READ NEXT INPUT RECORD.
CALL RECGET(CBUFF,LUN,IEOF,IC,MAXLEN,MARK)
RETURN
END

THIS ROUTINE READS A RECORD ON THE GIVEN LUN AND DETECTS END OF FILE

SUBROUTINE RECGET(CBUFF, LUN, IEOF, IC, MAXLEN, MARK)

DIMENSION CBUFF(140), CPBUFF(140), ICPBUFF(35)

INTEGER HARDWARE

CHARACTER CBUFF, CPBUFF, CHAR

900 FORMAT (13621)

DATA (ISHIFT=1)

DATA (LUNCOPY=0)

EQUIVALENCE (ICPBUFF, CPBUFF)

RETURN IF END OF FILE REACHED IN PREVIOUS CALL.

IF (IEOF.NE.0) GO TO 111

JUMP IF NOT TTY.

IF (HARDWARE(LUN).NE.6) GO TO 50

JUMP IF ASK COMMAND NOT ACTIVE.

IF (MARK) 50,40,15

PRINT OUT CONTENTS OF BUFFER.

DO 20 I=1,MARK

I=I+MAXLEN

ICCHAR=CBUFF(I)

CALL TTYSIGN(ICCHAR)

CONTINUE

IF (CBUFF(MARK+1).AND.CBUFF(MARK+2).NE.1R3) GO TO 50

CALL TTYSIGN(2R21)

GO TO 50

PRINT INFORMATION REQUEST SIGN ON TTY.

CALL TTYSIGN(1R+)

RESET BUFFER INDEX, READ RECORD, AND TEST FOR END OF FILE.

IC=1

READ (LUN, 900) (CBUFF(I), I=1, MAXLEN)

IF (EOF(LUN)) GO TO 100

CALL ROUTINE TO SEARCH INPUT STRING FOR EDIT CHARACTERS.

IST=0

CALL STREDIT(CBUFF, MAXLEN, IST)

JUMP IF LEFT ARROW (IGNORE CURRENT RECORD) EDIT CHARAC-...
C    CHARACTER FOUND.
01382  IF (IST.NE.0) GO TO 10
01383  IF (LUNCOPY.LE.0.OR.LUNCOPY.GE.64) RETURN
01384  IF (HARDWARE(LUNCOPY).EQ.1) RETURN
01385  IM=(MAXLEN+ISHIFT+3)/4
01386  DO 55 I=1,IM
01387  ICPUFF(I)=4H
01388  55 CONTINUE
01389  DO 60 I=1,MAXLEN
01390  K=I+ISHIFT
01391  IF (K.LE.0) GO TO 60
01392  CHAR=CP1UFF(K) =CBUFF(I)
01393  IF (CHAR.NE.1R ) LENC=K
01394  60 CONTINUE
01395  IF (LENC.LT.0) LENC=1
01396  CALL FWRITE(LUNCOPY,(LENC+3)/4,ICPUFF)
01397  RETURN
01398  100 IF(OF=1
01399  TC=MAXLEN+1
01400  LENC=2
01401  ICPFUFF=4H(EOF
01402  ICPUFF(2)=4H)
01403  GO TO 70
01404  ENTRY INPUTCOPY
01405  ISHIFT=EOF
01406  LUNCOPY=LUN
01407  RETURN
01408  END
01409
01410  THIS ROUTINE DETERMINES THE TYPE CODE FOR THE
01411  CHARACTER PASSED TO IT. THE CODES ARE -
01412  1 = LETTER, *, OR +
01413  2 = NUMERIC DIGIT
01414  3 = DECIMAL POINT
01415  4 = * OR -
01416  5 = BLANK
01417  6 = LEFT PAREN
01418  7 = RIGHT PAREN
01419  8 = EQUAL SIGN
01420  9 = COMMA
01421  10 = OTHER
01422
01423  FUNCTION ITYPE(CHAR)
01424  CHARACTER CHAR,CTAR
01425  DIMENSION CTAR(164),TAB(3)
01426  EQUIVALENCE (TAB,CTAR)
THIS ROUTINE CHECKS THE INPUT BUFFER CBUFF FOR THE
EDIT CHARACTERS BACKSLASH, AT SIGN, AND LEFT
ARROW AND PROCESSES THEM AS - DELETE PREVIOUS
CHARACTER (CUMULATIVE EFFECT OF MULTIPLE SLASHES)
IGNORE ALL CHARACTERS TO LEFT, AND IGNORE
ENTIRE RECORD RESPECTIVELY.
CBUFF = INPUT BUFFER

SUBROUTINE STEDIT(CBUFF, LEN, IST)
CHARACTER CBUFF, CBKUP, CKILL, CRECDELF
DIMENSION CBUFF(140)
CONTINUE
CKILL=1R4
CBKUP=1R0
CRECDELF=1RE
IST=0
J=1
DO 70 I=1, LEN
    IF (CBUFF(I), EQ, CRECDELF) IST=1
    IF (CBUFF(I).EQ, CKILL) GO TO 20
    IF (CBUFF(I).NE, CBKUP) GO TO 40
    J=J-2
    IF (J.LT.0) J=0
    GO TO 60
123
GO TO 60
STORE CURRENT CHARACTER Fetched BACK INTO BUFFER.

CBUFF(J) = CBUFF(I)
BUMP STORE INDEX.

J = J + 1
CONTINUE

BLANK FILL REMAINDER OF BUFFER.

DO 80 I = J, LEN
CBUFF(I) = 10
RETURN
END

THIS ROUTINE PRINTS AN ERROR MESSAGE ACCORDING TO THE CODE IN IERR, ON UNIT LUN.

SUBROUTINE PGERROR (IERR, SYM, ASYM, ICSTR)
DIMENSION IFST(24), IFMT(1), F(80), IBUFF(24), MSG(16)
DIMENSION ISYM(2), ITSYM(2)
EQUIVALENCE (ISYM, SYM), (ITSYM, ASYM)
EQUIVALENCE (F, IFMT), (X, IX)
INTEGER HARDWARE
DATA ((MSG(I), I = 1, 16) = 4H ERR, 4H ON , 4HKEYW, 4HORD , 4H;
4HWHI, 4HLE 3, 4HCANI, 4HNG , 4H, 4H AT ,
* 4HPOSII4HTION, 4H)
FORMAT (14)
DATA (IFST(01) = 1)
DATA (IFST(02) = 6)
DATA (IFST(03) = 10)
DATA (IFST(04) = 18)
DATA (IFST(05) = 24)
DATA (IFST(06) = 31)
DATA (IFST(07) = 39)
DATA (IFST(08) = 45)
DATA (IFST(09) = 50)
DATA (IFST(10) = 59)
DATA (IFST(11) = 65)
DATA (IFST(12) = 75)
DATA (IFST(13) = 82)
DATA (IFST(14) = 87)
DATA (IFST(15) = 97)
DATA (IFST(16) = 109)
DATA (IFST(17) = 119)
DATA (IFST(18) = 126)
DATA (IFST(19) = 137)
DATA (IFST(20) = 142)
DATA (IFST(21) = 155)
DATA (F(01) = 1HKEYWORD )
DATA (F(02) = 1HNOT DEFI )
DATA (F(03) = 1HNF NO K )
DATA (F(04) = 1HIELD CHAR )
DATA (F(07) = 1HACTER AF )
DATA (F(08) = 1HTER KEYW )
DATA (F(09) = 1HORDER NON )
DATA (F(10) = 1HINTEGER )
DATA (F(11) = 1H.uid CHAR )
DATA (F(12) = 1HCTER AF )
DATA (F(13) = 1HTER KEYW )
DATA (F(14) = 1HT 32767 )
DATA (F(15) = 1HOR ZERO )
DATA (F(16) = 1HESUBS CRIP )
DATA (F(17) = 1HET EXCEP )
DATA (F(18) = 1HARRAY )
DATA (F(19) = 1HBOUNDS )
DATA (F(20) = 1HVALUE MU )
DATA (F(21) = 1HST BE IN )
DATA (F(22) = 1HTEGER )
DATA (F(23) = 1HVALUE MU )
DATA (F(24) = 1HST BE RE )
DATA (F(25) = 1HABLE CAN )
DATA (F(26) = 1HABLE CAN )
DATA (F(27) = 1HOT BE )
DATA (F(28) = 1HESPECIFIC )
DATA (F(29) = 1HOD AGAIN )
DATA (F(30) = 1HVALUE MU )
DATA (F(31) = 1HST BE A )
DATA (F(32) = 1HSYMBOL )
DATA (F(33) = 1H )
DATA (F(34) = 1H )
DATA (F(35) = 1H )
DATA (F(36) = 1H )
DATA (F(37) = 1H )
DATA (F(38) = 1HCHARACTE )
DATA (F(39) = 1HR STRING )
DATA (F(40) = 1HT 32767 )
DATA (F(41) = 1HGEQ )
DATA (F(42) = 1HST SIG N )
DATA (F(43) = 1HOT FOUND )
DATA (F(44) = 1HEDROR IN )
DATA (F(45) = 1HVARDEF )
DATA (F(46) = 1HIVALI )
DATA (F(47)=8H0 TYPE )
DATA (F(48)=AH)
DATA (F(49)=8HINVALID)
DATA (F(50)=8HDEFINITI)
DATA (F(51)=8HNON OF AR)
DATA (F(52)=8HRAON ROUN)
DATA (F(53)=8HREGS IN VA)
DATA (F(54)=8HRDEF)
DATA (F(55)=8HANOTHER)
DATA (F(56)=8HVARIABLE)
DATA (F(57)=8HMUST BE)
DATA (F(58)=8HGIVEN F)
DATA (F(59)=8HIRST)
DATA (F(60)=8HUNABLE T)
DATA (F(61)=8H0 EQUIP)
DATA (F(62)=8HSAVED FI)
DATA (F(63)=8H2LE NUMP)
DATA (F(64)=8H2ER OF SU)
DATA (F(65)=8H3SCRIPT)
DATA (F(66)=8H2PECIFIC)
DATA (F(67)=8HED IS IN)
DATA (F(68)=8HVALE)
DATA (F(69)=AH)
DATA (F(70)=8HILLEGAL)
DATA (F(71)=8HLUN DAT)
DATA (F(72)=8HUM MUST)
DATA (F(73)=8H2PHA NUM)
DATA (F(74)=8H2EPIC OR)
DATA (F(75)=8HOTAL CO)
DATA (F(76)=8HINSTANT .)
DATA (F(77)=8HLE. 64)

C IF NO VALID OUTPUT LUN, DEFAULT TO 61
IF (LUN.LE.0) LUN=61

LUN=LUNA

GET START AND END INDEX OF ERROR MESSAGE.

II=IFST(IERR-2)
IJ=IFST(IERR-1)

JUMP IF OUTPUT LUN INVALID.
IF (LUN.LE.0 OR LUN.GT.64) GO TO 30

BUILD ERROR MESSAGE.
SYM=SYM
BSYM=BSYM
MSG(5)=ISYMX(1)
MSG(6)=ISYMX(2)
MSG(11)=ISYMX(1)
ENTRY PGERRLUN
   IF (IERR.NE.0) LUN=IERR
   GET INTEGER HALF OF REAL VARIABLE.
   X=SYM
   IF (IX.NE.0) LUNA=IX
   RETURN
   END

SUBROUTINE EWRITE(LUN,LEN,IBUFF)
   DIMENSION IBUFF(1)
   INTEGER HARDWARE
   ICD=-0
   CALL PITE(LUN,ICD,LEN,IBUFF)
   TF (HARDWARE(LUN),EQ.9.AND.ICD,NF.4H - 1) ICD=0
   RETURN
   END

SUBROUTINE CONVERT(SYM,NETYPE,XVAR)
   THIS ROUTINE TAKES THE BCD NUMBER, LEFT JUSTIFIED
   IN SYM, AND RIGHT JUSTIFIES IT.
   DIMENSION SYM(3),X(3),C(24)
CHARACTER C
EQUIVALENCE (X,C),(VAR,IVAR)
FORMAT (11X,19)
FORMAT (E20.0)
CONTINUE
ASSIGN GLOBAL ARRAY SYMBOL TO LOCAL EQUIVALENCED ARRAY.
X = SYM
X(2) = SYM(2)
X(3) = SYM(3)
SEARCH FOR END OF BCD NUMBER, MARKED BY J.
J = 20
DO 10 I = 1, 20
IF (C(J) .NE. 1R) GO TO 15
J = J - 1
RIGHT JUSTIFY NUMBER.
K = 20
DO 30 I = 1, 20
IF (J .LE. 0) GO TO 21
MOVE DIGIT.
C(K) = C(J)
J = J - 1
GO TO 30
BLANK FILL LEFT OF MOVED DIGITS.
C(K) = 1R
K = K - 1
IF (NTYPE .EQ. 2) GO TO 40
DECODE (20, 901, X) IVAR
GO TO 50
DECODE (20, 902, X) VAR
XVAR = VAR
RETURN
END
IDENT     VARPROC

* THIS ROUTINE STORES A VARIABLE, THE FIRST PARAMETER, AT THE ADDRESS SPECIFIED BY THE SECOND PARAMETER. THE THIRD PARAMETER IDENTIFIES THE TYPE - 1=1 WORD, 2=2 WORD, 3=3 CHARACTER.

ENTRY VARSTOR
ENTRY IVARGET
ENTRY IGEADDR

STARS EQU *
X1 EQU 1
X2 EQU 2

VARSTOP UJP STARS
STT H91,X1
LDI VARSTOR,X1
TMI 3,X1
STI VSTRTN,X1

LOAD,I -3,X1
SCHA CHARSTOR
SWA WS1
SWA WS2
LOAD,I -2,X1
TAT X2

LOAD,A,X1
UJP *+1,X2
JUMP ACCORDING TO TYPE OF ITEM

CHARSTOR SCHA STARS
STA H81

WS1 UJP H91
JUMP FOR RETURN JUMP

WS2 UJP H91
JUMP FOR RETURN JUMP

431 ENI STARS,X1
RESTORE X1

VSTRTN UJP STARS

* THIS ROUTINE PLACES THE ADDRESS OF THE VARIABLE IN THE PARAMETER LIST IN THE A REGISTER (TO LINK AS A FUNCTION WITH A FORTRAN PROGRAM).

IGEADDR UJP STARS
LOAD,I *+1
LOAD *+2
INQ,S 1
STO *+1
UJP STARS

* THIS ROUTINE FETCHES DATA STORED AT THE ADDRESS STORED IN THE
VARIABLE PASSED TO THIS ROUTINE.

0047 *
0049ivarget ujp,i
0050 1oa
0051 ina
0052 swa
0053 loa,i
0054 swa
0055 aba
0056 varget
0057 end
0059 ident
0060 entry
0061 ext
0062 tystgn ujp
0064 td i
0065 tin
0066 sti
0067 loa,i
0068 asc
0069 ujp
0070 ujp
0071 rcd
tai
0072 asciiout
0073 to
0074 asciiout
0075 r1
0076 r1
0077 r1