ESTI FILE COPY

ESD-TR-70-274

ESD ACCESSION LIST
ESTI Call No. 70753
Copy No. 1 of 4 cys.

USER'S MANUAL
COBOL COMPILER VALIDATION SYSTEM

July 1970

ESD RECORD COPY
RETURN TO
SCIENTIFIC & TECHNICAL INFORMATION DIVISION
(ESTI), BUILDING 1211

DIRECTORATE OF SYSTEMS DESIGN & DEVELOPMENT
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

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

AD0711369

LEGAL NOTICE

When U. S. Government drawings, specificatians ar ather data are used far any purpose ather than a definitely related government procurement aperatian, the government thereby incurs na responsibility nor any abligation whatsaever; and the fact that the gavernment may have farmulated, furnished, or in any way supplied the said drawings, specifications, or other data is not ta be regarded by implicatian ar atherwise as in any manner licensing the halder ar any ather person or canveying any rights or permission to manufacture, use, or sell any patented inventian that may in any way be related thereto.

OTHER NOTICES

Do nat return this capy. Retain ar destray.

ESD-TR-70-274

USER'S MANUAL
COBOL COMPILER VALIDATION SYSTEM

July 1970

DIRECTORATE OF SYSTEMS DESIGN & DEVELOPMENT
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

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

FOREWORD

The COBOL Compiler Validation System (CCVS) Users Manual is intended as a reference manual for on-site operations.

The system was developed as a part of Project 6917 under Contract F19628-67-C-0424 for the Electronic Systems Division (AFSC) by Information Management, Inc., San Francisco, California 94111. The project monitor was Mr. Russell A. Meier, ESMDA. The work was performed during the period from August 1967 through January 1969.

This technical report has been reviewed and is approved.

William F. Heisler

WILLIAM F. HEISLER, Colonel, USAF
Director, Systems Design & Development
Deputy for Command & Management Systems

ABSTRACT

This technical report consists of detailed specifications for the use of the COBOL Compiler Validation System (CCVS). The system is designed to measure the compliance of a specific COBOL compiler against the American National Standards Institute standard COBOL (ANSI X3.23-1968). This report describes the card input formats, deck structures, tape requirements, test modules, and operator procedures required to use the system.

TABLE OF CONTENTS

Section		Page
I.	INTRODUCTION	1
II.	SYSTEM DESCRIPTION	3
2.1	The USA Standard COBOL	3
2.2	CCVS Design Criteria	6
2.3	CCVS Components	9
2.3.1	The CCVS Data Base	10
2.3.1.1	Environmental Data	10
2.3.1.2	Tests	11
2.3.2	The Population File Maintenance Program	11
2.3.3	The Selector Program	12
2.3.4	The Source Program Maintenance Program	12
2.3.5	The Test Programs	13
2.4	How the CCVS is Used	13
2.4.1	Initial System Preparation	13
2.4.2	Test Program Generation	14
2.4.3	Test Program Execution	15
2.4.4	Test Results Evaluation	17
III.	USAGE INSTRUCTIONS	18
3.1	Population File Maintenance Program	18
3.1.1	Preparation of Inputs	18
3.1.1.1	Environmental Data	18
3.1.1.1.1	Environment Header	19
3.1.1.1.2	Environmental Data Cards	21

Section		Page
3.1.1.2	Tests	21
3.1.1.2.1	Test Header	21
3.1.2	Results of Operation	24
3.2	Selector Program	28
3.2.1	Preparation of Inputs	29
3.2.2	Results of Operation	33
3.3	Source Program Maintenance Program	35
3.3.1	Preparation of Inputs	36
3.3.2	Results of Operation	39
3.4	Character Code Conversion Program	40
3.4.1	Preparation of Input	40
3.5	Test Program Output Analysis	42
IV.	OPERATING INSTRUCTIONS	46
4.1	Compilation and Execution	46
4.2	Population File Maintenance Program	49
4.2.1	Input-Output Assignments	49
4.3	Selector Program	50
4.3.1	Input-Output Assignments	50
4.3.2	Sort Control Fields	52
4.4	Source Program Maintenance Program	53
4.4.1	Input-Output Assignments	53
4.4.2	Preparation of Test Program Change Deck	54
4.5	Character Code Conversion Program	55
4.6	Test Programs	56
4.6.1	Program 10	57
4.6.2	Program 11	58

Section		Page
4.6.3	Programs 12-14	59
4.6.4	Programs 15-16	59
4.6.5	Program 19	59
4.6.6	Programs 30-44	59
4.6.7	Program 45	60
4.6.8	Program 46	60
4.6.9	Program 47	60

Section		Page
APPENDIX I	DROP CODE LIST	61
APPENDIX II	DROP CODE USAGE IN TESTS	66
APPENDIX III	INITIATING ENVIRONMENTAL DATA	77
APPENDIX IV	TEST DIRECTORY	89
APPENDIX V	PFM DIAGNOSTIC MESSAGES	133
APPENDIX VI	CONSIDERATIONS IN CREATING TESTS	135
APPENDIX VII	SYSTEM GENERATION SPECIFICATION	142
APPENDIX VIII	SAMPLE CONTROL CARDS FOR CHARACTER CONVERSION PROGRAM	144

LIST OF FIGURES

Figure		Page
1.	Schematic Diagram Showing the Structure of the USA Standard COBOL	4
2.	The Validation Process	16
3.	Environmental Data Card Format	78
4.	Environmental Data Cards for XYZ-8795	84
5.	Output Listing of Environmental Data for XYZ-8795	86
6.	Relationship Between User Input and PFM Envrionmental Output	87
7.	Sequence Numbering for Test Cards	136

LIST OF TABLES

Table		Page
1.	Drop Code Usage in Tests	60
2.	Drop Code Assignments by Test	71

SECTION I

INTRODUCTION

The purpose of this manual is threefold. First, it provides an introduction to the COBOL Compiler Validation System. This introduction is directed both to prospective users of the system and to those who are merely interested in its objectives and how they are achieved.

Second, it provides detailed instructions in the use of the CCVS in measuring COBOL compilers against USA Standard COBOL. Finally, it presents guidelines that may be useful to those responsible for system maintenance or extension.

The manual is divided into four parts. Part 2, which immediately follows this introduction, is a general description of the COBOL Compiler Validation System. As prelude it offers a brief description of how USA Standard COBOL is structured, then discusses the design criteria under which the system was developed, the functions performed by the system's components, and finally, the way in which the system is used in measuring the degree of compiler standardization.

Part 3 presents the details of how to use the system. It describes the control information needed for expanding the system in terms of both its environmental "knowledge" and its repertoire of tests, and for generating test programs. A concluding section discusses the meaning of the results obtained from executing a test program and how such results can be traced through the system to find their purpose and resolve suspected compiler faults.

One of the primary design objectives of this system is the ability to run on any computer with a COBOL compiler. This flexibility places a limit on the detail with which this document can discuss certain facets of the system. This is particularly true in the area of operating instructions. Because the programs generated by the

system will run on a variety of computer-compilers, this manual can only enumerate the kinds of information an operator might require and indicate where he might find more specific instructions for operating within a particular environment.

Part 4, therefore, discusses in very general terms the kind of information that operating systems will require in order to correctly compile and execute the system itself and the test programs it generates.

SECTION II

SYSTEM DESCRIPTION

The COBOL Compiler Validation System (CCVS) is designed to measure the extent of compliance of any COBOL compiler with the set of COBOL language elements specified in USA Standard COBOL. To obtain a measurement, the user indicates to the CCVS the composition of the COBOL language set that forms the baeis for a particular evaluation. In reeponse, the CCVS produces one or more COBOL source programs that contain a representative sample of statemente drawn from the specified set. This program (or set of programe) ie then compiled and executed. During execution, the results are compared with those that should obtain if the compiler had been implemented according to the standard.

In order to guarantee a common baee of knowledge among the readers of thie document, a brief deecription of USA Standard COBOL follows. This description is excerpted from Chapter 1 of X 3.4 COBOL Information Bulletin #9, pages 1-1 through 1-6a.

2.1 USA STANDARD COBOL

In order to represent more effectively the uses for which COBOL has been designed, the historical organization of COBOL specifications, that is, Identification Division, Environment Division, Data Division, and Procedure Division has been revised within the proposed standard. The new organization is oriented around a functional processing module (FPM) concept. A nucleus (for internal processing) and seven functional processing modules have been defined. These functional processing modules are Table Handling, Sequential Access, Random Access, Sort, Report Writer, Segmentation and Library.

The Nucleus has been divided into two levels. The lower level contains the elements required for basic internal processing and is a proper subset of the high level. Each functional processing module contains two or more levels. In all caees, the lower levels are proper subsets of the progressively higher levele within the same module. In addition, some

functional processing modules contain a null set as their lowest level.

This organization provides the flexibility necessary to tailor specifications in such a way that they will satisfy the requirements of a large variety of data processing applications. At the same time it provides the ability to determine with a greater degree of certainty than previously possible the standard elements that comprise a given compiler.

The requirements for an implementation of USA Standard COBOL can be seen by referring to Figure 1. A COBOL implementation that includes a specified level of each of the functional processing modules and of the Nucleus, implemented as defined in the standard, will be considered to meet the requirements for an implementation of USA Standard COBOL.

	FUNCTIONAL PROCESSING MODULES						
NUCLEUS	TABLE HANDLING	SEQUENTIAL ACCESS	RANDOM ACCESS	SORT	REPORT WRITER	SEGMEN-TATION	LIBRARY
2NUC	3TBL	2SEQ	2RAC	2SRT	2RPW	2SEG	2LIB
	2TBL		1RAC	1SRT	1RPW	1SEG	1LIB
1NUC		1SEQ					
	1TBL		null	null	null	null	null

NOTE: The shorthand notation used in this figure is identical to that used in the CCVS control cards. It consists of a number indicating the level's position within the hierarchy and a three-character mnemonic name.

Figure 1. Schematic Diagram Showing the Structure of USA Standard COBOL

As an illustration, the following definitions of minimum and full USA Standard COBOL are given:

> The minimum USA Standard COBOL is composed of the minimum level of each functional processing module and of the Nucleus. Because of the presence of null sets, the minimum standard consists of the low levels of the Nucleus, Table Handling, and Sequential Access. The full USA Standard COBOL ie compoeed of the maximum level of each functional processing module and of the Nucleus.

Throughout the USA Standard COBOL Specifications there are certain language elements that depend, for their implementation, on particular types of hardware components. The implementor specifies the minimum hardware configuration required for a given implementation of USA Standard COBOL and the hardware componente that it supports. Any language elements that depend on hardware components for which support is claimed must be implemented. Those language elements that are not implemented because of their dependence on hardware components whose support is not claimed for an implementation must be so specified and their absence will not render the implementation non-standard.

When a facility is provided that accomplishes the functions specified by any particular COBOL element, it may be unnecessary to include that particular element within the COBOL source program. If such an unnecessary element doee appear in the source program, it must be accepted by the compiler. However, if the element does not lead to the production of object code, no substitute statements shall be required within the COBOL source program to accomplish this function.

By the same token, the inclueion of language elements or of functions that are not a part of the USA Standard COBOL specifications will not render an implementation of USA Standard COBOL non-standard. This is true even though it may imply the extension of the list of reserved words by the implementor, and prevent proper compilation of some programs which conform to the Standard.

2.2 CCVS DESIGN CRITERIA

The CCVS has been designed to fulfill the following functional requirements:

1. Thoroughness: The tests developed for each standard module must explore a compiler and its generated code to an extent sufficient to give a high probability of accuracy to its evaluation. This objective has been accomplished by testing a particular function to:

   a. Exercise all its basic structural variants. Structural variants arise from the format of a particular statement or function, as modified by the applicable syntactical rules. For example, the format

   $$\underline{\text{VERB}}\ \text{operand-1}\ \left\{\begin{matrix}\underline{\text{XX}}\\ \underline{\text{YY}}\end{matrix}\right\}\ \text{operand-2}$$

   yields tha basic structural variants

   VERB operand-1 XX operand-2
   VERB operand-1 YY operand-2

   b. Test a sample carefully selected from the set of possible combinations of operand types available to the statement.

   c. Test several variants arising from the operation of the ellipsis (...).

It is clear that in the majority of cases, a complete test of each structural variant, with every possible combination of operand types would result in an inordinately large number of statements. Therefore, a high degree of judgement was exercised in choosing a representative subset of test statements. Several possible types of tests have been eliminated from consideration.

a. No tests are made of erroneous statements. For example, syntactically incorrect statements are not generated. Such tests would

result in failure to compile or in run-time errors, either of which would result in confusion. This is especially true in the case of run-time errors (e.g., CLOSEing a file that is not in open status) because of the difficulty of associating an error with a particular statement in a program with a number of errors.

b. Tests are not designed to indicate how a function is implemented. Thus, the CCVS does not attempt to distinguish between good (efficient) and bad (inefficient) implementations.

c. No testing of non-standard extensions to COBOL is made. There are three classes of extensions that are of concern:

   1) Valid COBOL features that are not specifically referred to by the standard (e.g., COMPUTATIONAL-n and the APPLY clause are valid COBOL features but are not included in the standard because they are hardware-related features).

   2) Manufacturer extensions that have no relation to the remaining COBOL language. (e.g., IBM's TRANSFORM verb.)

   3) Manufacturer extensions that are required in order to utilize a standard COBOL feature. (e.g., IBM's ORGANIZATION and RECORD KEY clauses). In the latter case, we will manually include such statements in test programs in order to exercise functions that otherwise could not be tested (e.g., Random Access). The Air Force will have the same facility for future test program modifications.

d. No test of the ENTER verb is made because of the obvious difficulties involved.

2. Openendedness: The user must be able to alter the population of test statements to conform to changes in the USA Standard COBOL. For this reason, the user is provided the ability to add tests to the population file, modify the content of existing tests, delete tests, and change the modules against which a test is selected.

3. Ease of use: The CCVS must be relatively easy to use now and should become easier to use in the future. As the COBOL Standard becomes widely used, it is expected that more and more compilers will adhere to its specification. Thus, future ease of use should not be compromised in order to expedite testing the system using current non-standard compilers.

In order to accomplish this objective, the population of tests has been designed without reference to current implementations; the standard was the only reference material used. Thus, a "standard" compiler would compile and execute those modules it implemented with no adjustment whatsoever. This is the condition that (hopefully) will exist in the future. In order to make it easy for the user to cope with current implementations that are far from standard, a source program maintenance feature is provided. This feature facilitates the addition, deletion or modification of test program statements in order to tailor a particular test program to a non-standard implementation.

Additional features that make the CCVS easy to use are:

a. A test case can be specified by a user who does not have a detailed knowledge of COBOL. However, the activities involved in obtaining a test program that executes and evaluating the test results require some degree of expertise. As noted above, these activities should diminish in time.
b. Test results are clearly marked when they do not correspond to the result expected. The tests are documented in such a way that the offending test and the part of the standard to which it applies can be quickly identified.
c. The system is thoroughly documented to facilitate future maintenance

d. A particular compiler may be measured against any combination of Functional Processing Modules (FPM's), because the CCVS will construct test programs containing a single FPM or any combination of FPM's up to and including full USA Standard COBOL. The selection of FPM's to appear in a particular test program is based on a single, user supplied control card.

4. Machine Independence: Both the validation system and the test programs it produces must be available to operate on any computer for which a COBOL compiler is available and to do so with a minimum of reprogramming or manual intervention. The only hardware requirement imposed by the CCVS is minimal Input-Output configuration. The system requires that environmental data be supplied for each compiler that will be validated. Once this information has been supplied to it, the CCVS automatically generates the Environment Division and other hardware-oriented entries required by a particular test program. The environmental data is available to the CCVS until the user chooses to delete it.

5. Extendability: The underlying design of the CCVS must be of sufficient generality to be applicable to the validation of the compilers of other languages against their standards.

It is evident that one criterion usually specified for computer systems - namely, efficiency - is missing from this list. Efficiency, in the case of the CCVS, is of relatively little importance, because it will not be run as a regularly scheduled program. Furthermore, its efficiency would have been exceedingly difficult to predict since it depends almost entirely on the design point of the compiler being tested.

2.3 CCVS Components

The validation system consists of three computer programs and a data base. One program, the Selector, operates on the data base to produce the test programs called for by the user. A second program, Source Program Maintenance, is available to modify the selected test programs to: 1) remove statements that do not compile or run

successfully, and 2) add statements that are necessary to test non-standard features or that support standard functions. The third program, Population File Maintenance, is available to modify the set of available test statements that comprise the data base, and to add, delete or change environmental data.

Thus, the CCVS is not a set of COBOL Compiler test programs, but rather, a system capable of generating a very large number of "tailored" test programs. A particular test program can be tailored by the user to fit the environment of a particular COBOL compiler, to include representative operations from any subset of FPM's and to exclude tests of certain functions that are known to be inoperative or not implemented in the compiler undergoing evaluation.

The compiler test programs themselves need exist only during the short period of compiler evaluation. At all other times, the individual tests that comprise these programs are resident in the Population File.

2.3.1 The CCVS Data Base

The data base or "Population File" contains two distinct kinds of information: environmental data and tests. Both types of information are carried as unblocked 80-character COBOL source card images. The content of the Population File is discussed in detail in 3.1.2.

Environmental Data

The COBOL Environment Division contains a number of implementor names and other implementation dependent information. To a lesser extent, the Data Division contains information of a similar nature. In order to make the specification of a test program as easy as possible, this environmental data is maintained in COBOL Source card form for every compiler of interest. Thus, when a test program is to be generated for, say, the B-8500, the user merely specifies the

computer-name to the Selector Program, and it then selects the Environment Division entries appropriate to modules included in the test, using the environmental data contained in the B-8500 table on the Population File.

Environmental Data is carried on the Population File in order by mnemonic computer-name. A set of data for a particular compiler-computer consists of an environment header card followed by a series of COBOL source entries.

2.3.1.2 Tests

A Test consists of one or more COBOL statements that exercise a particular COBOL function (e.g., a verb). These statements are surrounded by a set of supporting Data and Procedure Division entries. The general sequence of statements within a Test is as follows:

1. Source and Result fields.
2. Initialization procedures.
3. The actual statements that perform the test.
4. Statements that move the test name, actual result and expected result to a common work area, and perform the result analysis and output routine that is common to each test program.

Tests are carried on the Population File in test serial number sequence. Each test begins with a test header, followed by the source cards comprising the test.

The various system utility programs comprise a special category of tests. While functionally different, they reside on the population file in order to utilize environmental data in the same way as tests.

2.3.2 The Population File Maintenance Program

This program operates on the Population File, and enables the user to add, modify or delete both environmental data and tests.

The most frequent use of this program will be the addition of environmental data for new compilers and the deletion of data for compilers that are no longer undergoing tests. The modification of tests should occur only when the USA Standard COBOL changes. Such a change may require the addition of new tests, the deletion of existing tests, the shifting of tests from one level of a module to another or between modules, or some combination of these. These operations are available to users through this program.

2.3.3 The Selector Program

The Selector program performs three functions:

1. Using the compiler-computer name supplied by the user, locates the applicable environment data and saves it.
2. Using the user supplied specification, selects those tests appropriate to the standard modules to be tested, deletes those tests designated by the user, and obtains the Environment Division statements required by the selected tests from the data saved in Step 1. For convenience, the Population File Maintenance, Selector and Source Program Maintenance Programs are handled like tests in order to supply them with Environmental Data.
3. Places the resulting series of tests and supporting statements in the order required for compilation, after removing certain data items with identical descriptions. Operating system control cards can optionally be placed before and after the source deck.

2.3.4 The Source Program Maintenance Program

This program is used to modify the source tape of a test program either before its initial compilation or between compilations. Modification may be necessary because:

1. The user wishes to test one or more non-standard features known to be implemented on the compiler, or
2. One or more test statements did not compile correctly or, having compiled, caused the object program to end abnormally.

2.3.5 The Test Program

Each test program produced by the Selector program is a complete COBOL source program, ready for compilation. The exact composition of a particular program depends on the contents of the information that the user supplied to the Selector program.

Each individual test within a program audits a particular feature or element of the COBOL language by executing one or more procedural statements. The result of that execution is then compared to a pre-determined "standard" result by a support routine. A parameter supplied to the Selector determines whether all tests results, or only those that differ from standard, are displayed.

2.4 HOW THE CCVS IS USED

This section describes the general activities required on the part of the user to validate COBOL compilers using the CCVS. The activities are divided into four phases: (1) Initial system preparation, (2) Test program generation, (3) Test program execution, and (4) Test result evaluation.

2.4.1 Initial System Preparation

The CCVS is delivered on an IBM 360 loadable tape. Appendix G contains a description of how to generate the Character Code Conversion Program, the Selector Program, and the Population File.

The initial step in using the CCVS is the compilation of the Selector program on the user's computer. This program, as well as Population File Maintenance and Source Program Maintenance, is written in a subset of minimum USA Standard COBOL to insure, as nearly as possible, that it will compile into a useable program when processed by any COBOL compiler. The COBOL subset is described in Appendix 4 of the CEI Detail Specification, Part I (reference 3).

The Selector program is carried on the Population File and can be selected with all its Environment Division entries completed, providing another computer is available on which to make the selection. Alternatively, the user can manually complete the necessary entries. Once the three phasee of the Selector have been compiled,

the object programs must be interfaced with this implemantor's Sort Program (the Selector does not utilize the COBOL SORT verb) and Operating System.

The Selector program in its object form may now be used to generate the Population File Maintenance program and the Source Program Maintenance program.

The Population File will be delivered with the user's computer reflected in the environmental data section. Thus, no Environment Division entries are required prior to this compilation. After compilation, this program must also be interfaced with the Operating System.

Under normal circumstances, the Selector program will be compiled and run on a single 'base' computer, as will the Population File Maintenance program. These programs in their object form will be used to generate test program(s) and a Source Program Maintenance program for each compiler to be validated. It is unlikely that the Selector program itself need ever be compiled on any other computer. Furthermore, it seems prudent to limit access to the Population File Maintenance program and, hence, it too need seldom, if ever, be recompiled.

2.4.2 Test Program Generation

The first step the user takes when he desires to generate a test program is to determine whether the Population File contains environmental data for the implementation in question. This can be ascertained from the latest print-out from the Population File Maintenance program. If the data is not there, it is created in the manner explained in appendix C and placed on the file by a Population File Maintenance run, as explained in Section 3.1.

Next, the user must decide which functions and which level of each function he wishes to measure the implementation against. There are at least two ways of arriving at this decision:

1. The reason for evaluation may provide the criteria. For example, if the compiler has been named as part of the answer

to a Request for Proposal, the RFP may contain a list of the modules that must be implemented, or the issuing agency may have a standard requirement that provides this information.

2. The implementor may claim in his advertising the level at which his compiler is rated. The user may simply wish to verify this claim.

If neither of these ways is open, the user can resort to the "relaxation" method. That is, he may measure the compiler against full USA Standard COBOL, then successively reduce the requirements until he achieves a clean compilation. This, obviously, is a time consuming approach.

Finally, the user can review the implementor's COBOL manual and determine which elements in the modules to be tested are not available in this compiler. These elements can be identified to the Selector program for elimination.

The next step in preparing the Test Program is to run the Selector program to generate the Source Program Maintenance program and the specified test program(s) for the implementation being evaluated. Thia run is discussed in Section 3.2.

Finally, a set of Operating Syatem control cards must be prepared according to the implementor's manual. These cards will be used for both the compilation and the execution of the program(s).

2.4.3 Test Program Execution

The validation process is diagrammed in Figure 2. The first step is the compilation of the source deck of the test program(s) as it emerges from the Selector Program. If the compilation is free of serious error messages, the object deck of the test program is executed. If compile-time errors have occurred, the user must trace each measage back to the source statement that

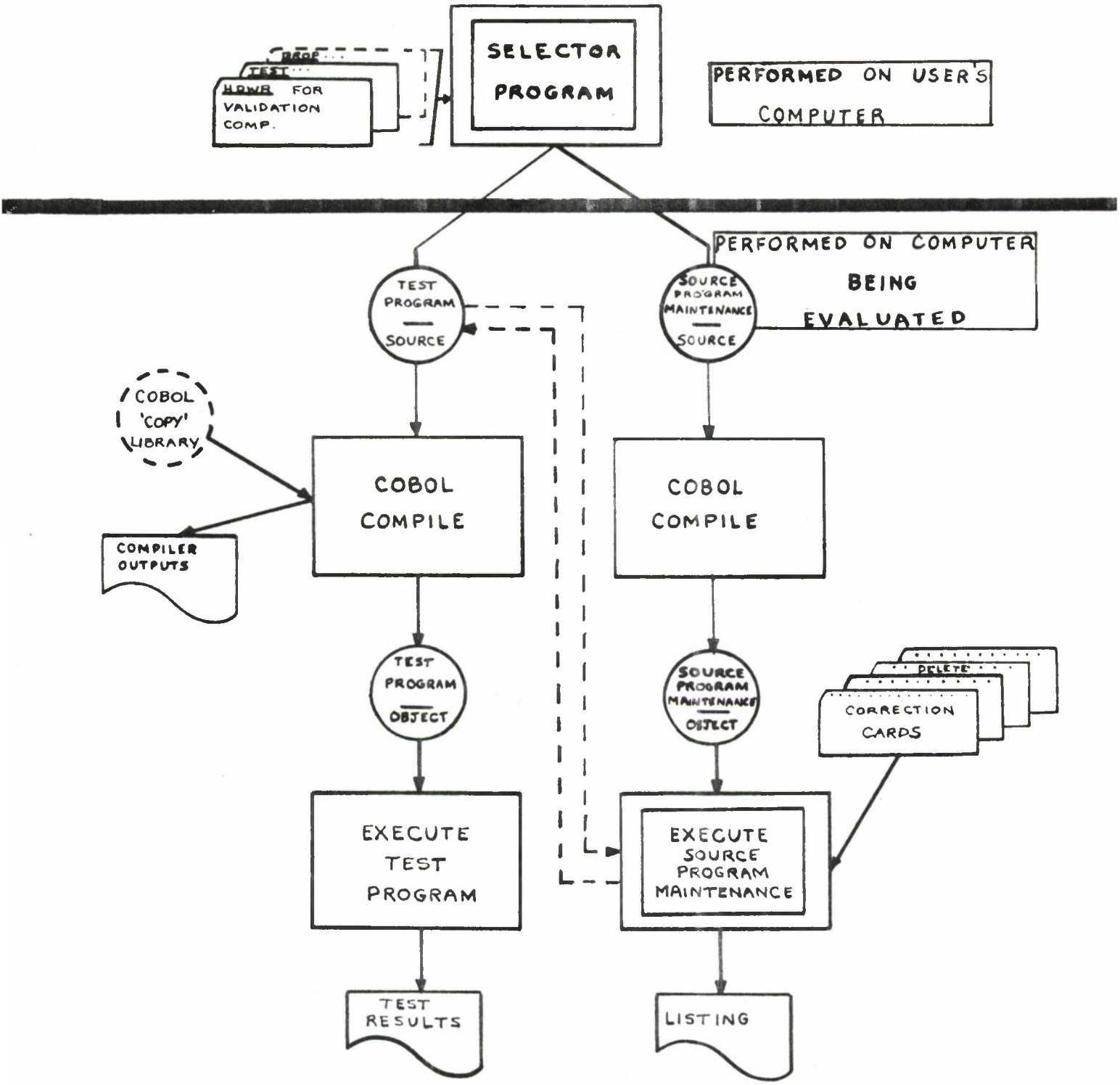


Figure 2. The Validation Process

caused it and, using the Source Program Maintenance program, modify the source deck of the test program to eliminate the errors. Output from the Source Program Maintenance program should be kept to document the changes. The test program is compiled (and modified as required) until all serious error messages are eliminated.

The final step of the validation process is to run the object version of the test program. If an error occurs during the run, the error must be traced to the source program, the program must be modified using Source Program Maintenance, then compiled again.

2.4.4 Test Results Evaluation

The result of a validation run - the answer to the question of compiler compliance with a particular set of Standard modules - must be determined from:

1. The list of language elements initially eliminated by the Selector program.
2. Any messages from the Selector program concerning missing environmental data.
3. The modifications made to the source deck of the test program as indicated in the output of Source Program Maintenance.
4. The actual test results that are flagged in the output of the test program. The test codes that identify each result can be used to locate in the Test Directory the test that produced the questionable result. This Directory indicates the module to which the test applies and the particular feature being examined.

SECTION III

USAGE INSTRUCTIONS

3.1 POPULATION FILE MAINTENANCE PROGRAM (PFM)

The PFM program maintains the CCVS data base - the Population File. This file contains two distinct kinds of data: Environmental Data and Tests. Both types are composed of groups of related COBOL source entries, with the groups sequenced by an identifying code. The PFM treats each type of data separately. It has the facility to add, delete, change the content of, or merely print groups of each type. Input to PFM consists of entry information arranged in the same sequence as the Population File, and the Population File itself. Output is a revised Population File, error messages, and listings of the groups added, deleted, changed or requested for printing.

3.1.1 Preparation of Inputs

3.1.1.1 Environmental Data

Each unique configuration for which a test program is to be generated is identified to the CCVS by an indicative name and is represented on the Population File by a set of COBOL source entries. These source entries are constructed by the PFM from user supplied input and can be referred to by any test in such a way that the Selector Program will replace the reference by the entry referred to. In this way, the tests and the system utility programs are tailored to run on an individual configuration.

Environmental input consists of a header card that contains the indicative computer name and an indication of the action to be performed, followed by one or more data cards (unless a "delete" action is specified).

3.1.1.1.1 Environment Header: The format of the Environment Header card is shown below.

```
0 0                          1
1 3                          3
E indicative-computer-name  {A}
                            {C}
                            {D}
                            {P}
```

The Environment header card is identified by the 'E' in column 1, followed by a blank in column 2.

Columns 3 through 12 contain the indicative-computer-name by which the PFM, the Selector, and the user refer to the entries. Because environmental data groups are sequenced by indicative name and because collating sequence differs among computers, a consistent format for this name must be adopted by each user.

Indicative name is assigned by the user and has the basic fixed format:

AAANNNNNNN

where:

AAA contains the alphabetic abbreviation for the configuration's manufacturer. No spaces are permitted. For example, Control Data Corporation could be abbreviated as CDC, General Electric as GEC and Burroughs as BUR. Any abbreviation can be used as long as it is three letters long with no imbedded blanks.

NNNNNNN is all numeric and contains the configuration's model number and any other information such as system number, compiler level, etc. While the only requirement is that this field be numeric, it is suggested that the last two

digits be used to guarantee uniqueness among different configurations of the same series computer.

Column 13 contains the Action Code. Four options are available:

A Add a new set of entries identified by the indicative name, to the Population File. When this option is used, all twelve environmental data cards must follow the header.

D Delete the set of entries identified by indicative name. When this option is used, no environmental data cards follow the header.

C Change the set of entries to reflect the information contained in whichever data cards follow. The header is followed by from one to twelve environmental data cards. Replacement is done on a card by card basis. Therefore, all entries in a given card must be filled out correctly, even though only one entry represents a change. For example, if entry 31 on card 9 is to be changed, entries 26 through 30 and 32 through 34 must be filled out exactly as they were for the initial entry, even though they themselves are not being changed. (see Figure 3, Appendix C for card format).

P Print the entire set of environmental entries for indicative name. Because an A, D or C option will cause automatic printing of the entire set of entries, the indicative name appearing with a P option must not appear anywhere else in the environment input stream for a given PFM run.

3.1.1.1.2 Environmental Data Cards. There are twelve environmental data cards that can follow the environment header when the operation specified is "add" or "change". These cards are filled out according to a set of questions using information from the implementor's COBOL manual.

Appendix C contains the questionnaire, a blank environmental data sheet showing the card formats, and a sample environmental data sheet for a fully implemented USASI compiler together with the corresponding output of a PFM run.

3.1.1.2 Tests

Each Test begins with a test header that contains the Test's serial number, the program number in which the test appears, the module membership indicators, and one or more drop codes that, when specified to the Selector program, cause the Test to be deleted from a particular test program. The header is followed by the source cards that comprise the Test. Each card of a Test is coded with a unique sequence number that is used by the Selector program to place the card in the correct sequence for compilation.

3.1.1.2.1 Test Header. The format of the test header is shown below.

```
1 3                    8  9                11        55
T test-serial-number { A } program-number  modules   drop-codes
                     { C }
                     { D }
                     { P }
```

Columns 1 and 2 of the Test Header contain T followed by a space. Columns 3 - 7 contain the test serial number. Tests are carried on the Population file in test serial number sequence. This serial number uniquely identifies each test to the PFM, Selector and the user, and has the format:

```
n { Nmmm }
  { XXmm }
  { YYmm }
```

where n is the level (1, 2 or 3) within the module; N identifies the Nucleus and mmm the sequence of the test within the Nucleus; XX identifies one of the functional modules and mm is the sequence number of the test within that module; or YY identifies a non-test resident (e.g., a system utility program) and mm is the phase number or 01 if the resident has only one phase.

The functional module identifiers are:

TH	Table Handling
SQ	Sequential Access
RA	Random Access
ST	Sort
RW	Report Writer
SG	Segmentation
LB	Library

The current non-test resident identifiers are:

PF	Population File Maintenance
SL	Selector
SP	Source Program Maintenance
SU	Support Routine

Column 8 contains the following action identifiers:

A Add the test header and the series of cards between it and the next header to the Population File.

D Delete the test whose serial number begins in Column 3 of this card. When this option is used, no source test cards follow the header.

C Change the test identified in columns 3 - 7 to reflect the data in the cards that follow. This header replaces the existing header. If source cards follow the header, they are merged into the existing test in order by sequence number (a new card replaces an old card with the same sequence number).

P Print the test. Since options A, D and C cause the test to be printed, this option is used to print tests not otherwise mentioned in the input stream for this PFM run.

Columns 9 - 10 contain the program number in which the test appears. Program numbers have been assigned as follows:

01	Support Routine (never appears alone)
02	PFM
03	SPM
04	Selector I
05	Selector II
06	Selector III
07-09	Reserved for system programs
10	NUC (except 1N304, 1N305, 1N314, 1N315, 2N050, 2N051 and 2N052)
	TBL
	SEQ (except RERUN, 1SQ27-31, and 2SQ15)
	RAC (except 2RA18)
	2SRT
	1RPW (except 1RW01)
11	1SEQ (RERUN, 1SQ27-31)
	2RPW
12	1SEG
13	2SG01
14	2SG02
15-18	LIB (except 1LB04, 1LB06)
19	1NUC (CURRENCY SIGN, 1N314; DECIMAL POINT, 1N315)
20	1N304, 1N305, 2N050, 2N051, and 2N052
21	2RA18
22	2SQ15
23-29	Unassigned
30-44	1SRT (1 test per program)
45	1LB04
46	1LB06
47	1RW01
48-49	Unassigned
50	Library Entries for 1LIB, 2LIB
99	9ZZ99 (header only -- for artificial end-of-file)

Columns 11 through 54 contain from 1 to 9 designations of the module and level to which the test applies. All tests in the low level of a module designate both the low and high levels. The

designation is coded in standard form separated by commas with no intervening spaces (e.g., 1TBL,2TBL,3TBL).

Columns 55 - 80 contain up to 9 two-digit "drop codes". These codes refer to COBOL language features that are commonly not implemented in currently available compilers. The present list of codes is found in Appendix A. This list can be expanded by the user by merely adding features and corresponding unique codes to the list. During a Selector run, when a drop code appears in a test header and a Selector program DROP card, the test is dropped. The codes are separated by commas with no intervening spaces (e.g., 1D,2E,5C).

3.1.2 RESULTS OF OPERATION

The Population File that is produced by this run is unlabeled and consists of unblocked 80-character card images. The sequence of information on the file is as follows:

1. Environmental Entries, in sequence by the indicative-computer-name in columns 3 through 12 of the Header card of each set. An environmental set consists of:
   a. The header
   b. Fifty-two environmental data cards, in sequence by columns 1 through 6.

2. Tests and other residents, in sequence by the test serial number in columns 3 through 7 of the Header card of each test. A test consists of:
   a. The header
   b. Any number of source cards in sequence by columns 1 through 6.

The Population File contains the following items:

1. Environment sets:

   BUR0350001
   CDC0640001
   GEC0062501
   IBM0036001
   UNV0110801

2. Tests and other residents:

0LB00		
0SL01	through	0SL03
0SU01		
1LB01	through	1LB10
1N001	through	1N31B
1RA01	through	1RA0B
1RW01	through	1RW03
1SG01	through	1SG14
1SQ01	through	1SQ31
1ST01	through	1ST15
1TH01	through	1TH04
2LB01	through	2LB10
2N001	through	2N061
2RA01	through	2RA18
2RW01	through	2RW05
2SG01	through	2SG02
2SQ05	through	2SQ15
2ST01	through	2ST06
2TH01	through	2TH04
3TH01	through	3TH06
9PF01		
9SP01		
9ZZ99		

Test 9ZZ99 consists of a header only and is used by the Selector program as a machine-independent end-of-file indicator.

The result of a PFM run depends on the action specified:

1. Add Action. The add action in the case of the Environmental data creates a header and 52 card images. The add action for a Test places the card images complete with header on the population file exactly as they are received from the input device. The new entries are printed out.

2. Delete Action. The delete action in either the case of the environmental data or the tests deletes all card images on the population file until the next header is found. In both cases the deleted information will be printed out.

3. Print Action. The print action simply prints and copies the card images up until the next header.

4. Change Action. The change action in the case of the Environmental data changes only that information on a particular input card. The sample below will illustrate this:

The input cards:

```
E XYZ0879501C
06                    READ-UNIT
```

will change ET0230 and ET0240 under the header E XYZ0879501C from (See Appendix C for an explanation of Environmental cards):

```
ET0230                DISK2
ET0240                READER
to
ET0230
ET0240                READ-UNIT
```

Note that the blank field on the input card was transmitted to ET0230 which means that all information generated by an environmental data card must be present if it is not to be blanked out.

The change action for a test operates on the basis of the serial numbers in columns 1 through 6 on the test card images. Equal serials cause replacement by input, unequal serials cause collation in sequence. The following example will clarify:

The input cards:

```
T 2ST01C102SRT                                        4G ,4U

854400      ADD 1
854405       TO SUP-NUM-WK.
```

will change

```
854400      ADD 1 TO SUP-NUM-WK.
854410      MOVE '*' TO SUP-CTL (SUP-NUM-WK).
```

to

```
854400      ADD 1
854405       TO SUP-NUM-WK.
854410      MOVE '*' TO SUP-CTL (SUP-NUM-WK).
```

and the old header

```
T 2ST01A102SRT                                        4G ,4T
```

will change to

```
T 2ST01C102SRT                                        4G ,4U
```

In both the test and the environment case the entire set of card images between headers of the changed file will print out.

Appendix E contains the list of diagnostic messages produced by PFM.

3.2 SELECTOR PROGRAM

The Selector generates test programs from Population File entries, based on user supplied control information.

The Selector is capable of supplying any resident on the Population File with environmental data. The only provisions are that: 1) the resident must be entered through a PFM run in the manner specified for tests, and 2) the resident must be named on the TEST card. The PFM, SPM, the Selector itself and 'COPY' Library entries (for tests of LIB) are currently supplied with environmental data in this fashion.

The inputs to the Selector are the Population File and a set of control statements that define the test programs that are to be generated.

Principle considerations in the preparation of these control cards are the machine-compiler configuration subject to test, the modules of USA Standard COBOL against which the compiler is to be measured, which (if any) standard features are known not to be implemented, and the operating system's requirements for the subject machine and the compiler environment. Each consideration is expressed in a separate control card.

The Selector produces one or more test programs plus a set of diagnostic and informative messages.

3.2.1 Preparation of Inputs

The Selector accepts five control cards, the first two of which are required. They are as follows:

HDWR - Hardware description: This card is required and indicates the indicative-name assigned to the environmental data to be used by the Selector in tailoring the test programs. The format of the HDWR card is:

```
1     6
HDWR  indicative-name    ,ALL
```

Indicative-name must be identical to the name that appears on the header card of an environmental data set (see 3.1.1.1.1). This environmental data will be used to tailor whatever Population File residents are named on the TEST card.

The optional parameter ALL specifies that all test results are to be printed by the test programs when they are executed. The omission of this parameter causes only those results that differ from the expected results to be printed.

TEST - test specification card: This card is required and indicates: 1) the content of the test program in terms of Standard modules, and 2) which CCVS utility programs (or data) are to be selected from the Population File and tailored. More than one TEST card can be used if a single card is overflowed. The format of the TEST card is:

```
1     6
TEST  nXXX  ,nXXX  ...
```

The indicator nXXX may be:

1. The name of a USA Standard COBOL module (e.g., 1NUC, 2RPW). Appendix D discusses the tests that comprise each module.

2. The special indicators:

   a. 1MIN which stands for the minimum standard requirement - 1NUC, 1TBL, 1SEQ. When 1MIN is specified, modules other than the three for which it stands may also be specified.

   b. 1MAX which stands for the maximum standard requirement - 2NUC, 3TBL, 2SEQ, 2RAC, 2SRT, 2RPW, 2SEG, 2LIB. When 1MAX appears, no other modules may be specified.

3. The resident designators:*

OSL1	Selector, phase 1
OSL2	Selector, phase 2
OSL3	Selector, phase 3
OLIB	"COPY" Library entries for both 1LIB and 2LIB
1PFM	Population File Maintenance program
1SPM	Source Program Maintenance program

*O equals zero

In the table below, an X indicates which combinations may appear together on a TEST card:

	1MIN	1MAX	nNUC	nTBL	nSEQ	other modules	residents (not tests)
1MIN						X	X
1MAX							X
nNUC				X	X	X	X
nTBL			X		X	X	X
nSEQ			X	X		X	X
other modules	X		X	X	X	X	X
residents	X	X	X	X	X	X	X

If the computer on which the test program is to run has a small main storage capacity, it is advisable to limit the number of modules that appear on a TEST card. Multiple Selector runs (not multiple TEST cards in a single run) will then be necessary to generate all the test programs needed to satisfy the particular validation requirements. If a further reduction in test program size is required, the DROP card can be used to specify by serial number certain tests to be eliminated during selection.

<u>DROP - Drop indicators</u>: The Drop Card is optional and specifies one or more items to be eliminated during the selection process. This enables the user to eliminate features that he knows are not implemented in the compiler being validated before the test program is selected. For example, in validating most currently available COBOL compilers for 1TBL, the Indexing feature would be DROPed. More than one DROP card can appear in the input stream.

```
1      6
DROP   identifier    ,identifier  ...
```

Identifier may be either:

1. A two digit "drop code". The relationship of Drop codes to standard COBOL features is shown in Appendix A. A single drop code may cause one or a number of tests to be dropped from one or more of the modules specified on the TEST card. Up to 75 codes may be specified for a Selector run.

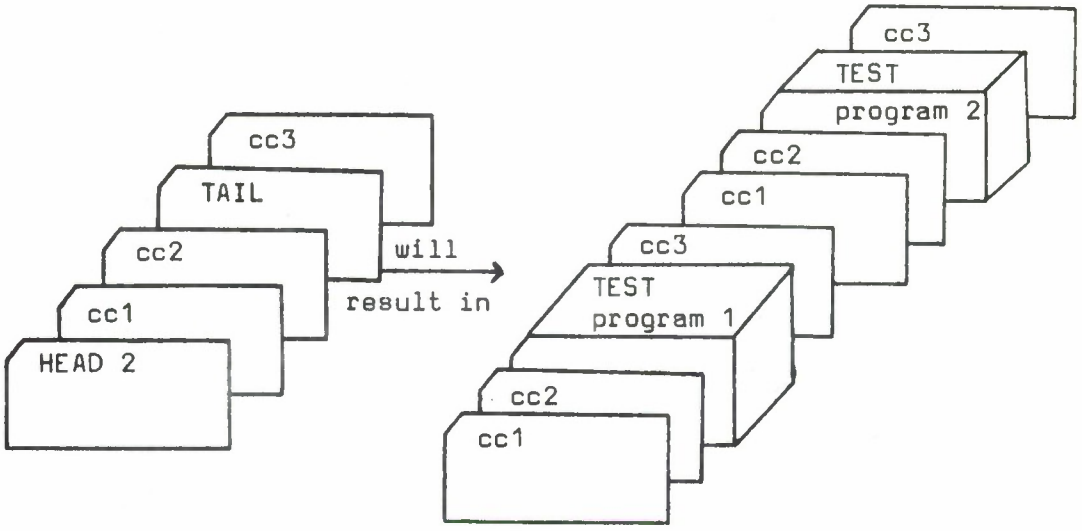
2. A five character test serial number. Each test has been assigned a unique serial number (see 3.1.1.2.1) that associates the various statements comprising the test. This serial number is found in the Test Directory (see Appendix D) and when used in the DROP card, causes all the statements related to the particular test to be dropped. Up to 50 serial numbers can be specified for a Selector run.

It should be noted that if a particular test contains both implemented and non-implemented statements, the SPM program should be used to modify it rather than using a drop code to eliminate it, unless the entire test is rendered meaningless by the absence of the non-implemented statements.

<u>HEAD and TAIL Cards</u>: These optional cards enable the user to place Operating System Control cards before (HEAD) and after (TAIL) the source deck of each test program. This feature may enable a smoother run sequence to be obtained under certain Operating Systems. The format of these cards is:

```
1      6
{HEAD} nn
{TAIL}
```

where n is the number of control cards. This number of cards must follow the HEAD or TAIL card in question. For example:



Only one HEAD and one TAIL card may appear in the input stream; each may contain a different number, but the sum of the two numbers must not exceed 50.

The control cards must be recognizable as such to the Selector program; that is, they must not contain end-of-file indications, nor can they contain all numeric data in columns 1-6.

As shown in the example above, when the composite of modules selected on the TEST card results in the generation of more than one test program (see 3.1.1.2.1), the control cards are repeated and placed before and after each individual source program.

3.2.2 Results of Operation

The Selector generates one or more test programs depending on the parameters of the particular TEST card. The assignment of modules to programs is described in 3.1.1.2.1. The selected programs appear in program number order on the output file.

In addition, the Selector produces an information and diagnostic listing. This list contains two kinds of information in the same format:

1. A list of the serial numbers of all tests dropped together with the cause of their removal and the modules in which the tests normally appear.

2. A list of non-standard situations reflected in the environmental data for the particular implementation. For example, the lack of a RESERVE clause would be reported at this point.

This listing contains three columns: (1) Test serial number, (2) message, which contains a short message or a drop code, and (3) the module and level to which the test belongs, taken from the test header card (see 3.1.1.2.1). For example:

SERIAL	MESSAGE	MODULE
1N004	4N	1NUC,2NUC
1N316	4B	1NUC,2NUC
	4T	
1TH02	1M	1TBL,2TBL,3TBL
2SQ11	ENV ERR FATAL	2SEQ
2TH02	1M	2TBL,3TBL
3TH01	NAMED ON DROP	3TBL

The first three tests were eliminated because their test headers contained a drop code identical to one contained on the DROP card. Notice that the second test, 1N316, was dropped by two drop codes. The fourth test was dropped because the environmental data it requested was missing, and the card in the test that requested this data indicated that the entire test must be dropped in this event (see Appendix F for further details). The fifth test was dropped by the same drop code, 1M, that dropped the third test. The last test was dropped because its serial number appeared on the DROP card.

The lack of environmental data requested by a particular test results in one of 2 additional messages:

ENVIRONMENT HAS NO ENTRY...CARD DROPPED
ENVIRONMENTAL ERROR CAUSES FOLLOWING TEST TO DROP...

These messages are interspersed with those described above.

3.3 SOURCE PROGRAM MAINTENANCE PROGRAM (SPM)

The primary purpose of SPM is to permit source level modifications to test program contents. Modifications may be required if language elements are not implemented in the compiler undergoing tests, or if that compiler observes conventions different from or in addition to those observed in USA Standard COBOL.

Many of these differences can be handled more easily through the Drop Code facility or by entries on the Environmental Data portion of the Population File. Character punch code differences are handled by the Character Code Conversion Program described in 3.4.

Any differences that cannot be handled by those means, but which will have an effect on the execution of the tests, must be handled by means of an SPM run. For example, a compiler might require certain non-standard information in FD entries. These entries cannot be provided through the environmental data facility and, thus, must be added by means of SPM.

Since the complete universe of changes which may become necessary cannot be predicted, no set working procedure can be established. Some factors to consider during test preparation for a specific compiler include these:

1. Does the implementor note any non-standard features, or extensions to USA Standard COBOL that are necessary for the correct operation of standard features?
2. Does the manufacturer note any restrictions, un-implemented or partially-implemented features?

3. Is the implementation on a par with the level of each module to be validated in the tests?

4. Are there known problem areas or "bugs" in the current version of the compiler?

5. Does the compilation of the test program produce error messages, or is the execution incorrect?

6. Do system considerations have an effect on the test design (e.g., file passing techniques)?

If the answers to any of these questions indicate that changes to the test programs are required, the following points must be considered in making these changes.

1. How many tests are affected by the change?
2. Does the change invalidate the purpose of any individual test?
3. Can the change be made more easily by use of either the environment table (see PFM, 3.1) or Selector control cards (Selector, 3.2)?
4. Does the change require reprogramming or redesign of any test?

3.3.1 Preparation of Inputs

Input to an SPM run consists of the file of test programs produced by the Selector, and a set of change cards. Change cards must be in sequence by program number (columns 73-74), then sequence number (columns 1-6).

SPM considers any card that is not in sequence by program number/ sequence number to be a system control card. Because system control cards do not usually contain fields for sequence number or program number, changes to these cards cannot be made by SPM. Rather, the source program must be punched out and such changes made manually. The system end-of-file (whatever is recognized as an AT END condition by the COBOL READ statement) cannot appear in the source or maintenance inputs except to designate the end of such inputs.

All modifications are made on a card-by-card basis. To make changes to individual words within a statement - for instance, to delete an optional word from the middle of a statement - the entire card on which that statement appears must be replaced by one containing the desired wording. The individual tests have been designed in such a way as to facilitate card-by-card modification.

The SPM input card formats and corresponding functions are as follows:

<u>OPTION</u>: This card controls the content of the SPM output listing. An OPTION card applies to all programs following it until overridden by another OPTION card.

1	7	13	14	73	74
	OPTION	L		program-number	

1-6: must be blank.

7-12: OPTION.

13: L or blank. L causes the listing of the output source program with the changes annotated (this option is assumed if no OPTION card is found by SPM). Blank causes the listing of correction cards only.

14: must be blank.

73-74: program number or blank.

DELETE: Single source entries or a series of source entries may be deleted by means of this card.

```
1              7    13              18          73   74
sequence-no-1DELETEsequence-no-2            program-number
```

where sequence-no-1 identifies the first or only source program card to be deleted and sequence-no-2 identifies the last card to be deleted. When only one card is to be deleted, sequence-no-2 must be equal to sequence-no-1. Columns 73-74 must contain the program numbers.

No DELETE card may indicate a range which overlaps the range of another DELETE card.

Insertions: These require no control card. The cards to be inserted must be assigned serial numbers in columns 1-6 which fall between a pair of existing serial numbers.

When a group of cards is to be inserted at a single point, only the first card need be punched with a serial number; the remaining cards may be blank in columns 1-6. However, all must contain the program number in columns 73-74.

Insertions may be made within the range of a DELETE.

Replacements: Replacements are similar to insertions except that the serial number field of the replacing card equals the serial number on the source program card to be replaced.

When a series of cards contains more cards than are being replaced, the extra cards may either be assigned serial numbers between the last replaced one and the following one; or, as with inserts, contain a blank sequence field.

When a series of replacement cards contains less cards than are being replaced, the excess must be deleted by a DELETE control card. In this case, it may be more convenient to DELETE the entire group to be replaced, following it by an insertion. Note that blank serial numbered cards following from a DELETE card are permitted.

END:

1	7	73 74
	END	program-number

1-6: blank
7-9: END
10-72: blank
73-74: program-number or blank

SPM input cards are arranged in the following order:

1. OPTION card (optional).
2. Program groups; arranged in sequence by program number (columns 73-74).
   a. OPTION card (optional).
   b. Deletions, insertions and replacements in order by sequence number (columns 1-6). Card with blank sequences numbers are treated as if they were consecutively numbered from the first card of their group.
3. END card.

3.3.2. Results of Operation

An SPM run results in: (1) an updated source tape of test programs, (2) a listing for each program modified by the run, as specified by the option card in force when the program was processed.

3.4 CHARACTER CODE CONVERSION PROGRAM

Because of the differing representation of character codes in different computers, a character code conversion program has been furnished. This program, unlike the other CCVS utility programs, is not written in COBOL. Rather, it is a System/360 BAL program that operates as a free-standing program.

The program accepts any COBOL source program or the Population File as input, together with two control cards specifying the source and target character codes. Output of the program is the converted program or Population File.

3.4.1 Preparation of Input

Control cards required by the program are:

1. SOURCE card:

   a. Columns 1 - 6 = SOURCE.

   b. Columns 7 - 9 = the address of the input device; i.e., 180, 00C, etc.

   c. Columns 10 - 11 = If 2540 is used, 01 = Data Mode 1, 02 = Data Mode 2. If tape is used, the mode setting shown in the chart below should be used.

   d. Columns 12 - 80 = Characters to be translated.

2. OBJECT card:

   a. Columns 1 - 6 = OBJECT.

   b. Columns 7 - 9 = the address of the output device; i.e., 180, 00D, etc.

   c. Columns 10 - 11 = If 2540 is used, 01 = Data Mode 1, 02 = Data Mode 2. If tape is used, the mode setting shown in the chart below should be used. For all other situations, leave these columns blank.

d. Columns 12-80 = Characters to which SOURCE character are to be translated. Each OBJECT character should appear in the same column as that SOURCE character from which the translation is to be made.

Col 10-11	Bytes per Inch	Parity	Translate Feature	Convert Feature
10	200	odd	off	on
20	200	even	off	off
28	200	even	on	off
30	200	odd	off	off
38	200	odd	on	off
50	556	odd	off	on
60	556	even	off	off
68	556	even	on	off
70	556	odd	off	off
78	556	odd	on	off
90	800	odd	off	on
A0	800	even	off	off
A8	800	even	on	off
B0	800	odd	off	off
B8	800	odd	on	off
C0	800 1600	single-density 9-track tapes only dual-density 9-track tapes only		
C8	800	dual-density 9-track tapes only		

A set of sample control cards is found in Appendix H.

The following rules must be observed when determining device assignments.

1. If either the input or output device is a card reader that will be reading or punching other than EBCDIC punches, that device must be a 2540 with the data mode 2 option.

2. When either input or output is assigned to tape, only EBCDIC punches may be used in either control card.

3. Remember that, on tape, the objective is a particular bit configuration; the punches in the control cards should be selected with this in mind.

3.5 TEST PROGRAM OUTPUT ANALYSIS

The ALL parameter of the HDWR card conditions a test program to display the actual and pre-determined result of every test in a test program. If an actual and a pre-determined result differ (as determined by a character-by-character comparison) the actual result is underlined. The absence of the ALL parameter results in the display of the actual and pre-determined results of only those tests where a difference is found.

Test results are displayed acroes the output page. Four lines appear:

1. The top line contains the test serial number (e.g., 1N026, 1TH02).
2. The second line contains the pre-determined result.
3. The third line contains the actual result.
4. The fourth line contains dashee that underline those cases wherein the pre-determined and actual results do not agree.

If a failure is noted in the output, it is necessary to trace the error to its cause in order to determine whether or not the failure represents a violation of the USA Standard COBOL. The procedure to follow in gathering information about a particular test ie as follows:

1. Note the test serial number which appeare in the first line of the test result printout.
2. Look up that test in the test directory found in Appendix D. This will indicate what the purpose of the test is and how the test results are related to that purpose.

3. If more detailed information is desired, look up the test in the test descriptions found in Part II of the Detailed Program Specification (Reference 4).

4. Look up the test on the listing produced by the PFM. This is the ultimate source of information on the procedures and design of the test.

Once the source of the difference is isolated, it must be considered in light of the purpose of the test. As an example, suppose a sort test indicates a single error, as follows:

```
1 S T 0 9           *
1 1 1 1 1 1 1 1 1 *
1 1 1 0 1 1 1 1 1 *
- - - - - - - - - *
```

The presence of dashes in the last row indicates an error somewhere in the results. Comparing the middle two rows shows the error to be in the fourth position. The first row identifies the test as 1ST09.

Looking at the test directory under 1ST09, we see that each position represents the most recently changed sort key field on consecutive sorted records and that the fourth record in the ascending sequence expected has, as its most recently changed field, the eighth key with a generated value of -14.

Going to the test listing for 1ST09, we note that the eighth key is computational, unsigned. Thus, the expected result that was checked for was +14.

This indicates that an unsigned computational item which originally had been assigned to a value of -14 had failed when used as a sort key.

The error must now be correlated to a feature. Check over other results to see if any other failures are related to this one. If so, try to determine the trouble by isolating the most common cause. Check all diagnostics produced at compilation and at execution, and the manufacturer's literature for restrictions, special implementations or known problems.

If the error can be correlated to a feature other than those being tested in the failing test, the test should be modified to eliminate it:

1. Determine the change required in order to bypass the problem without invalidating the purpose of the test.
2. Prepare the change using SPM change techniques. The program number in which the test appears can be found in the test header on the PFM list. The sequence numbers found there for the test source cards are the same as those found on the test program listing.
3. Using the SPM make the changes to the test program (<u>not</u> to the tests themselves) and re-execute. In the case of data items, the item to be changed may appear under a different test serial number because of the elimination feature of the Selector Program (see 2.3.3 and reference 4).

As a continuation of the earlier example, all test results are checked for failure related to sort keys, computational or unsigned fields. Several failures of unsigned computational fields might be found in tests of another module indicating a problem with such fields. If a diagnostic message or a manufacturer's manual warns of computational items being considered as always signed, it could be assumed that this is the cause of the failure.

Since the purpose of test 1ST09 is to test certain RETURN statement situations, the use of this particular key is incidental and thus the test would still be valid without it.

Several changes might be made to eliminate the error from the test but perhaps causing the expected value check to be changed from +14 to -14 would be the easiest.

In the teet program listing for 1ST09, card 851180 (Columns 1 - 6) is found to contain the 14 ueed in the existing check. A replacement card 851180 with -14 is prepared and used in an SPM run to update test program 38. The SPM output provides a record of the change.

SECTION IV

OPERATING INSTRUCTIONS

4.1 COMPILATION AND EXECUTION

Because the CCVS is intended to be usable with any COBOL compiler, it is readily apparent that it must operate in any of a large number of environments. For this reason, the compilation and execution of the test programs are discussed here in general terms, rather than in relation to any particular implementation.

Several items will be useful to the operator in preparing to run the CCVS:

1. The implementor's COBOL manual,
2. The appropriate operating system manual, (a COBOL related abstract may be contained in the COBOL manual),
3. A Printout of test program(s) from Selector run (or Source Program Maintenance).

The paragraphs that follow outline a series of steps that must be taken to prepare for compilation and execution.

Determine if the implementation being tested will allow stacked input, i.e., multiple programs not separated by end-of-file marks. If not, the user must take one of the following approaches:

1. Choose tests such that only one program will be produced by the Selector program.
2. Punch out the Selector output file and manually separate the programs, using columns 73-74.
3. Write a program to copy the Selector output file and insert the appropriate end-of-file marks between programs.

Determine the method of assigning "implementor-name" (in the ASSIGN clause of the SELECT sentence) to a physical device. This is often done at execution time with a type-in or control card, although in some systems it may be specified either

explicitly or implicitly in the ASSIGN clause itself. If neither Sequential nor Random Access is being tested, the user need only determine the assignment for the devices to be used for printing (often a printer, although sometimes a file that is printed in a separate operation), and for ACCEPT and DISPLAY statements (these may not require explicit assignment).

If Sequential without mass storage is being tested, the tape and printed "implementor-names" must be assigned to their appropriate devices. If either Sequential mass storage operations or the Random Access module is being tested, the mass storage "implementor-names" must also be appropriately assigned.

Determine any unusual control cards that may be necessary. For example, a machine that has no hardware switches may simulate them via control information.

If Segmentation is being tested, determine whether any control cards are needed or affected thereby.

If Sort tests are being tested, extra control information may be needed by the sort program invoked by the Sort statement.

Some determination must also be made about what is necessary to invoke execution of both the compiler and the resulting object program. Implementations differ considerably in this area, ranging from simple to very complex.

Although the preferred procedure is to set up the job as a "compile and go" operation, as nearly self-contained as possible, an occasional implementation may not allow this approach.

There are a number of areas in which machine dependent problems may manifest themselves. For example, the bit configuration generated by arithmetic on signed fields is sometimes not represented by a printer graphic. When this occurs, extra care is necessary to insure that proper interpretation is made.

Some machines have rather specialized input media such as paper tape or magnetic tape only. When this occurs, some means of translation from punched card format must be devised. It should be remembered, however, that a machine of a particular make and model can usually be found somewhere with provision for punched card format input.

A situation sometimes arises in which input-output software limitations cause problems. For example, compilers occasionally limit references to any given FD to only one mode (i.e., INPUT, OUTPUT, etc.). This is a severe limitation that renders the compiler non-standard. To run Sequential and/or Random Access module tests with this restriction, the user must run on an individual test basis using the "DROP" facility in the Selector Program and possibly the Source Program Maintenance Program to construct the runs.

Another non-standard restriction is more easily circumvented in the test runs but could be quite cumbersome in a production environment. This is the case where the "implementor-name" in the ASSIGN clause of the SELECT sentence is restricted to representing a unique physical device. (e.g., tape unit 1). To run the tests in this environment, a separate physical device is required for every FD in a given run and the "implementor-names" inserted by the Selector have to be changed using the Source Program Maintenance Program.

In the sections that follow, some specified considerations such as input-output assignments and file-passing requirements, are discussed for each program in the CCVS.

4.2 POPULATION FILE MAINTENANCE PROGRAM

4.2.1 Input-Output Assignments

The following table defines I/O requirements for this program:

File Name	Printer Input Output	Usage
PØP-FILE	i	Old Population File master.
TAPE-ØUT	o	New Population File master.
CØNTRØL-CARD	i	Update cards.
PRINTER-ØUT	o	Listing.

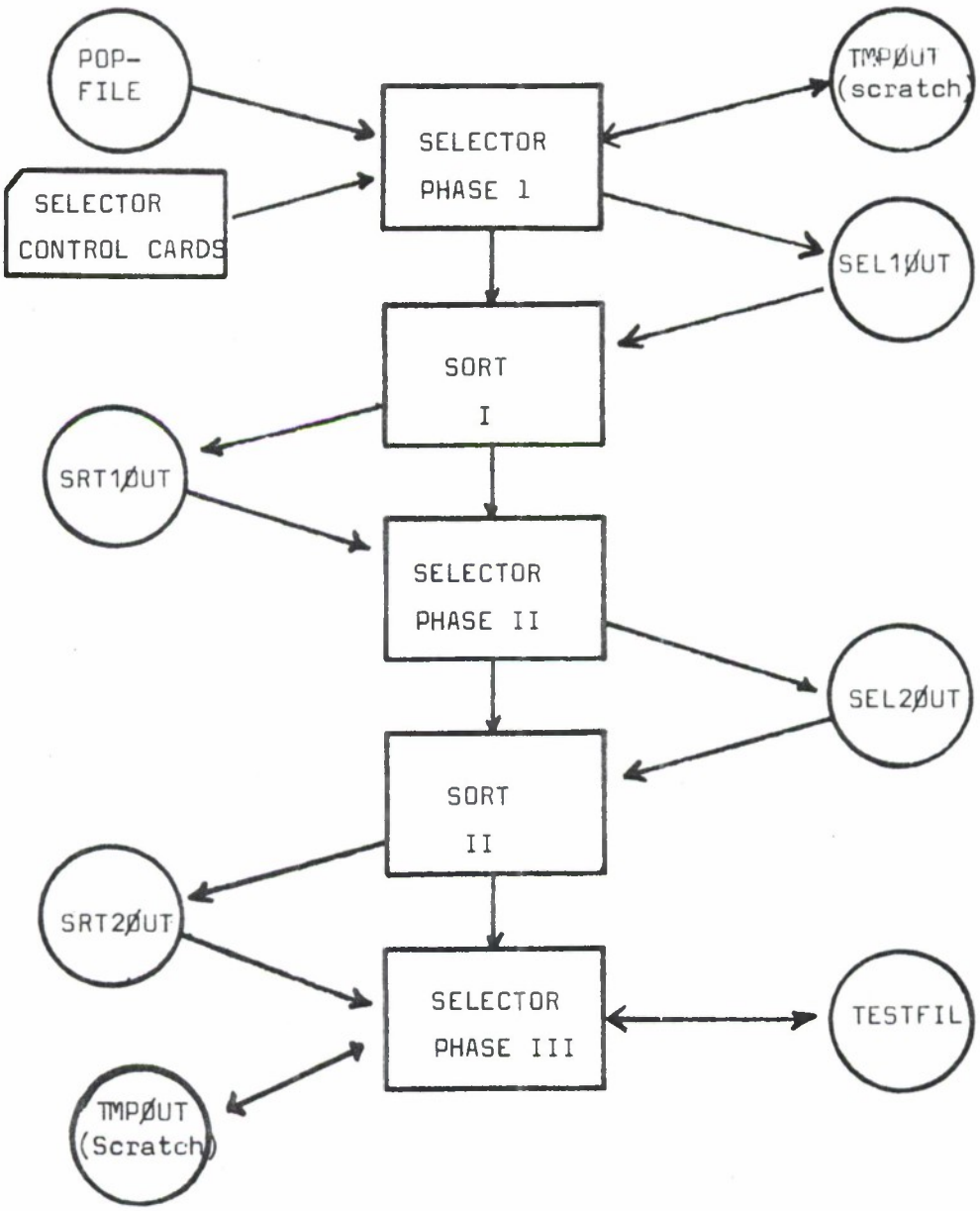
NOTE: Ø indicates alpha O in file-names.

4.3 SELECTOR PROGRAM

4.3.1 Input-Output Assignments

The Selector consists of three COBOL programs which must be interfaced with two sorts. The sorts, either system sorts or simple COBOL sort statements of the USING, GIVING type, must be provided as separate jobs or job steps. File passing must also be provided for between the execution of these phases.

A typical job flow using a system control-card type of sort facility is shown below.



Files used by the Selector are:

PHASE	Name on chart	FD Name	printer input output scratch	file passing	record size
Sel I	Population File	PØP-FILE	i	not passed	80
Sel I	Selector Control File	CØNTRØL-CARD-FILE	i	not passed	80
Sel I	TMPOUT	TEMP-ØUT-FILE	s	not passed	80
Sel I	SEL1OUT	SEL-ØUT	o	passed	117
Sel I		PRINTER-ØUT	p		
Sort I	SEL1OUT		i	not passed	117
Sort I	SRT1OUT		o	passed	117
Sel II	SRT1OUT	SEL-INS-FILE	i	not passed	117
Sel II	SEL2OUT	SEL-ØUT-FILE	o	passed	88
Sel II		PRINTER-ØUT	p		
Sort II	SEL2OUT		i	not passed	88
Sort II	SRT2OUT		o	passed	88
Sel III	SRT2OUT	SEL-INS-FILE	i	not passed	88
Sel III	TESTFIL	SEL-ØUT-FILE	o	passed	80
Sel III	TMPOUT	SEL-TMP-FILE	s	not passed	88
Sel		PRINTER-ØUT	p		

All messages are placed on PRINTER-OUT. Messages include certain error checking plus sequence checking. In the event of an error in the sequence of records coming from a sorting step, the selector requests on the printer that the sort be performed again and then terminates the phase.

4.3.2 Sort Control Fields

The Selector appends keys to the front of the records to be sorted, and arranges the key fields such that the entire key may be used as a single key for simplicity. The relative sequence of records with like keys is unimportant and no assumptions are made by the Selector based upon such sequences. Since records are not modified during sorts, input record counts should equal output record counts.

Sort I

Major key	1-2	Alphanumeric	Ascending
Inter key	3-32	"	"
Minor key	33-37	"	"
(body)	38-117		

Sort II

Major key	1-2	Alphanumeric	Ascending
Minor key	3-8	"	"
(body)	9-88		

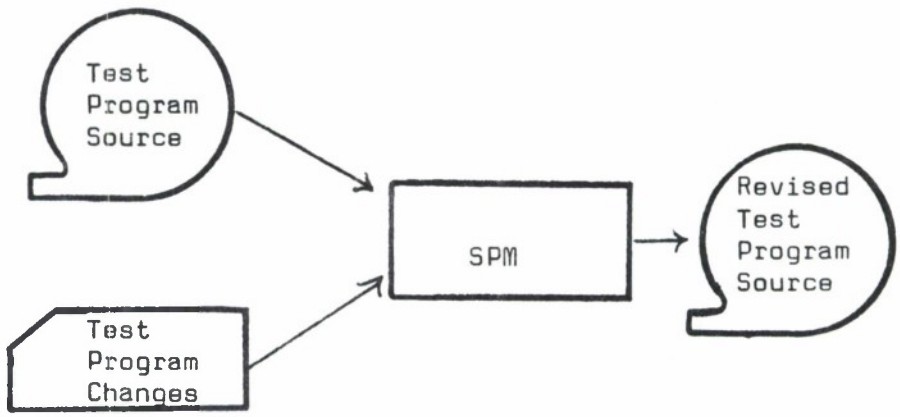
4.4 SOURCE PROGRAM MAINTENANCE PROGRAM

4.4.1 Input-Output Assignments

SPM is a single phase program whose purpose is to permit updating a test program file by means of a maintenance file in order to form a new test program file.

The operator must verify the control card set up for the maintenance file and for the compile and execution (or just execution if previously compiled) of SPM itself. The source program for SPM is obtained by use of the Selector Program.

A typical job flow is:



The table that follows defines the I/O requirements:

Name on chart	FD Name	printer input output	Record Size	Notes
Test-Program Source	SØURCE-FILE	i	80	
Test-Program Changes	MAINT-FILE	i	80	May be card reader, tape or disk.
Revised Test Program Source	ØUT-FILE	o	80	
	PRINT-FILE	p		

4.4.2 Preparation of Test Program Change Deck

System control cards may be included to provide an end-of-file for this deck. The deck may be pre-placed on a tape or disk file or it may be read from a card reader.

If the END (column 7-9) card is used as the last card in the change deck, no attempt will be made to read beyond that card.

The order of the deck is sequence number (column 1-6) within program-number (column 73-74), with these exceptions:

1. OPTION (column 7-12) cards may appear at the front of the deck.
2. END (column 7-12) card may appear at the end of the deck and serves the same function as an end-of-file.
3. Change cards may be blank in columns 1-6 if they follow a change or a DELETE card which has a value in column 1-6.

4.5 CHARACTER CODE CONVERSION PROGRAM (Sec. 3.4)

The CCVS Character Code Conversion program is an IBM System 360 BAL program and may be run on any System/360 with a console typewriter with unit address 01F, a card reader with unit address 00C, and the additional I/O devices needed by the user.

1. Place object deck followed by the "SOURCE" and "OBJECT" control cards, followed by the input deck (if any) in the card reader hopper.
2. Ready any other devices to be used.
3. Press LOAD. An END OF JOB message will be typed out when the run is completed.
4. I/O errors and control card errors will be typed out in the following format:

```
I/O ERROR ddd STATUS aaaa SENSE bbbb
     WHERE:  ddd = device address
            aaaa = status bytes
            bbbb = sense bytes
INTERVENTION REQUIRED ddd
JOB CANCELED - PERMANENT I/O ERROR ddd
JOB CANCELED - CONTROL CARD ERROR cc
     WHERE:  cc = error code
```

CODE	ERROR
10	CC1-6 do not contain "SOURCE" or "OBJECT"
20	CC7-11 do not contain valid hex characters (0-9,A-F)
30	CC10-11 do not contain a legal data mode

4.6 TEST PROGRAMS

The Selector program is capable of generating a number of test programs. The exact number generated for any particular run depends on the content of the TEST card. A epecific test program, in turn, may or may not contain the teste for a particular module. Thus, each test program preeente ite own set of operational probleme, but the number of these to be reeolved during a given teet eeseion is variable.

All teet programe utilize the output device identified ae the PRINTER in the environmental data. Other devices used depend on the exact content of the program. In most caeee, the control information required by a particular implementation can be obtained from the source listing of the test program. Some unueual problems are discussed below, by program number.

4.6.1 Program 10

Program 10 contains, at most, tests of: NUC, TBL, SEQ, RAC, 2SRT, and 1RPW (except 1RW01).

SEQ and RAC present no particular problems. 2SRT interfaces through the SORT verb with the implementor's generalized sort program. This interface may cause additional control information requirements:

1. Some control systems require special information on files utilized by the USING and GIVING options of the SORT. The following file-names describe files associated with one of these two options:

   FILE-NAME-USING-2SRT-1
   FILE-NAME-USING-2SRT-3
   FILE-NAME-GIVING-2SRT-4

2. Sort work files may not necessarily be ASSIGNed in the Environment Division, but rather, may have implicit assignments that require that control information be submitted for prescribed device-names. This will usually be mentioned in the implementor's COBOL manual.

RPW produces output that may have to be printed off-line. The exact format of tapes written under control of a particular implementation's Report Writer feature must be determined from the COBOL manual. Of particular interest are: (1) record length, (2) block length, (3) relative position and meaning of the printer carriage control symbols used by Report Writer, and (4) recording mode of the report tapes.

LIB presents a special problem because the input format of entries to the implementor's "COPY" library and the methods for creating and maintaining such a library are not specified in the USA Standard COBOL.

Library entries are carried in the Population File in program 50 and can be acquired for a particular implementation by specifying TEST OLIB. Each entry is preceeded by a header that contains the name by which the entry is referred to by the COPY statement in the test program. This header is merely for information and is not for entry in the library.

The library entries must be placed in the "COPY" library in the format and using the method of the implementation in question. Both items of information can usually be found in the COBOL manual.

4.6.2. Program 11

Program 11 contains, at most, tests of 1SEQ (1SQ22), and 2RPW. The general considerations discussed in program 10 apply to program 11 as well.

2RW03 tests the CODE clause of the RD by interspersing three reports on a single file (FILE-NAME-REPORT-2RPW-3). The COBOL Report Writer assigns a unique code to records of each of the three reports. Presumably, the implementor supplies a utility program that is able to separate the file at print time. If such a program is not available, the tape can be dumped for visual inspection.

4.6.3 Programs 12-14

Programs 12-14 contain tests for SEG. These tests have no explicit Input-Output requirements. Some implementations may require additional control information because one or more object program segments reside on an external device. Because it is impossible to predict how many segments will be forced onto external storage, (if any), this information will have to be found in the source program listing, in a procedure map or similar display.

4.6.4 Programs 15-18

Programs 15-18 contain the library tests (except 1LB04 and 1LB06). The OLIB library entries must have been placed in the "COPY" library prior to compiling programs 15-18.

4.6.5 Program 19

Program 19 contains the tests 1N314, CURRENCY SIGN and 1N315, DECIMAL POINT. Neither test requires any control information.

Program 20

Program 20 tests DISPLAY without the UPON option, then ACCEPT without the FROM option, followed by ACCEPT...FROM, then DISPLAY ...UPON. The devices addressed by these statements may have to appear in an entry on an operating system control card.

The DISPLAY statement of 1N304 outputs the string

AB.....YZ + - > < = $, ; . ( ) / ø 01.....9

on the DISPLAY device. If this device is visible to the operator, this display will indicate that ACCEPT will immediately follow.

The ACCEPT statement of 1N305 requires as input the character string:

ABCDE.....XYZ012.....9 + - *

with no intervening spaces. If the ACCEPT device is the operator's console, the operator must key in this string at the appropriate time. Otherwise, the string must be entered on the media used by the ACCEPT device (card-reader, paper-tape reader, etc.).

The ACCEPT statement of 2N050 (which immediately follows the ACCEPT of 1N305) requires as input the character string.

123456789*

The DISPLAY from 2N051 produces the character string:

2N051 ABCDEFGHIJ0123456789

The final DISPLAY from 2N052 produces the character string:

2N052 -AB...YZ + - > < = $, ; . ( ) / ø 01.....9

2N052 -AB...YZ + - = $, ; : ( ) / ø 0.....9A...Z + - = $, ; . ( ) / ø 0...9

Programs 21 and 22

Programs 21 and 22 test Declarative LABEL PROCEDURES but present no problems.

4.6.6 Programs 30-44

Programs 30-44 each contain one test of a 1SRT feature. The individual programs generally alternate in function between creating a file to be sorted, and performing the sort itself and verifying the output. As a result, files must be passed from the creating programs to the program that sorts and verifies. The implementor's techniques for identifying files passed between programs must be used.

The table below indicates the files passed by the various programs:

FD file name	created in	Used in	Rec Size	Notes
FILE-NAME-USING-3	31	32 (USING)	18	
FILE-NAME-USING-4	32	33 (USING)	18	
FILE-NAME-GIVING-4	33 (GIVING)	34	18	
FILE-NAME-GIVING-6	35 (GIVING)	36	27	Alt. device possible
FILE-NAME-USING-11	39	40 (USING)	50-100	differing lengths
FILE-NAME-USING-11	40 (GIVING)	41	50-100	differing lengths
FILE-NAME-USING-14	42	42 (USING)	33	multi-reel
FILE-NAME-GIVING-14	43	44	33	multi-reel

4.6.7 Program 45

Program 45 is identical to 1ST01 except that the SD entry is copied from the "COPY" library.

4.6.8 Program 46

Program 46 is identical to 1RW02 except that the RD entry is copied from the "COPY" library.

4.6.9 Program 47

Program 47 contains 1RW01 as it is not compatible with other programs. The report writer considerations mentioned under program 10 apply to program 47 also.

APPENDIX I

DROP CODE LIST

The COBOL language features that may be dropped during Selection are listed under DESCRIPTION. The column headed FPM shows the Functional Processing Module to which each feature belongs.

DESCRIPTION	FPM	CODE
LANGUAGE CONCEPTS		
Punctuation Character ,	2 NUC	1A
;	2 NUC	1B
Relation Characters > <	2 NUC	1C
=	2 NUC	1D
Condition-names, level-88, Condition-name test	2 NUC	1E
Procedure-names, all numeric	2 NUC	1F
Figurative Constants: plural forms	2 NUC	1G
ALL, except for 'character'	2 NUC	1H
Logical connectives AND OR, Compound conditions	2 NUC	1J
Qualification: Data Division	2 NUC	1K
Procedure Division	2 NUC	1L
Indexing feature	1 TBL	1M
Reference Format: continuation of words and numeric literals	2 NUC	1N
IDENTIFICATION DIVISION		
DATE-COMPILED	2 NUC	2A

DESCRIPTION	FPM	CODE
ENVIRONMENT DIVISION		
CONFIGURATION SECTION		
SPECIAL-NAMES CURRENCY SIGN	1 NUC	3A
DECIMAL-POINT	1 NUC	3B
I-O SECTION		
OPTIONAL Files	2 SEQ	3C
RESERVE clause	2 SEQ	3J
SAME AREA	1 SEQ/RAC	3N
RECORD option	2 SEQ/RAC	3P
SORT option	2 SRT	3R
series option	2 SEQ/RAC/ SRT	3S

DESCRIPTION	FPM	CODE
DATA DIVISION		
Level numbers: 66	2 NUC	4S
88	2 NUC	1E
single digit	2 NUC	4A
Abbreviations SYNC, JUST, PIC, COMP	1 NUC	4B
BLOCK integer-1 TO option	2 SEQ	4D
CODE	2 RPW	4E
GROUP INDICATE	2 RPW	4F
JUSTIFIED RIGHT	1 NUC	4G
LABEL RECORDS data-name	2 SEQ/RAC	4H
OCCURS		
ASCENDING/DESCENDING	3 TBL	4J
INDEXED BY	1 TBL	1M
DEPENDING option	3 TBL	4L
PICTURE mixed A, X, 9 in AN Picture	1 NUC	4M
AN edited items	1 NUC	4N
Currency Sign	1 NUC	3A
Decimal Point	1 NUC	3B
RECORD CONTAINS	1 SEQ/RAC	4P
RENAMES and level 66	2 NUC	4S
SYNCHRONIZED	1 NUC	4T
USAGE COMPUTATIONAL	1 NUC	4U
INDEX	1 TBL	1M
VALUE literal series	2 NUC	4V
literal THRU literal	2 NUC	4W
VALUE OF data-name IS data-name	2 SEQ/RAC	4X

DESCRIPTION	FPM	CODE
PROCEDURE DIVISION		
Arithmetic formulas	2 NUC	5A
Conditions: Relational operators > <	2 NUC	1C
=	2 NUC	1D
Condition-name condition	2 NUC	1E
Compound, logical ops. AND, OR	2 NUC	1J
Abbreviation 1	2 NUC	5B
2	2 NUC	5C
Options: ROUNDED	1 NUC	5E
SIZE ERROR	1 NUC	5F
Multiple result fields-Arith. verbs	2 NUC	5G
CORRESPONDING, ADD and SUBTRACT	2 NUC	5H
NO REWIND, OPEN and CLOSE	2 SEQ	5K
Verbs and options		
ACCEPT FROM	2 NUC	6F
CLOSE UNIT	2 SEQ/RAC	5L
LOCK	2 SEQ/RAC	5M
COMPUTE	2 NUC	5P
DISPLAY UPON	2 NUC	6H
DIVIDE BY option	1 NUC	5R
REMAINDER option	2 NUC	5D
IF nesting on true path	2 NUC	5S
nesting on false path	2 NUC	5T
MOVE CORRESPONDING	2 NUC	5U
OPEN REVERSED	2 SEQ	5V
PERFORM VARYING 2 and 3 levels	2 NUC	5W

DESCRIPTION	FPM	CODE
READ INTO	2 SEQ/RAC	6J
RELEASE	2 SRT	5X
RETURN INTO	2 SRT	5Y
SEARCH (format 1)	3 TBL	5Z
ALL (format 2)	3 TBL	4J
SEEK	1 RAC	6A
SET	1 TBL	1M
USE		
Error procedure	2 SEQ/RAC	6B
Label procedure	2 SEQ/RAC	4H
BEFORE REPORTING	2 RPW	6C
WRITE ADVANCING		
BEFORE	1 SEQ	6D
AFTER	1 SEQ	6E
FROM	2 SEQ/RAC	6K

APPENDIX II

TEST - DROP CODE CROSS REFERENCE

TABLE 1. DROP CODE USAGE IN TESTS

CODE	FPM	USED IN TESTS
1A	2NUC	2N061
1B	2NUC	2N061
1C	2NUC	2N043
1D	2NUC	2N024-2N032,2N034,2N035,2N043,2N045,2N046
1E	2NUC	2N059
1F	2NUC	2N061
1G	2NUC	2N058
1H	2NUC	2N058
1J	2NUC	2N039-2N042
1K	2NUC	2N058
1L	2NUC	2N053,2N054
1M	1TBL	1TH02-1TH04,2TH02-2TH04,3TH01-3TH04 ,2N022
1N	2NUC	2N008
2A	2NUC	2N060
3A	1NUC	1N314
3B	1NUC	1N315
3C	2SEQ	2SQ05
3J	2SEQ	2SQ06-2SQ08,2ST04

3N	1SEQ/RAC	1SQ11,1SQ12
3P	2SEQ/RAC	2ST02
3R	2SRT	2ST03
3S	2SRT	2ST03
4B	1NUC	1N316
4D	2SEQ/RAC	2RA01-2RA08,2SQ06-2SQ08
4E	2RPW	2RW03
4F	2RPW	2RW01,2RW02
4G	1NUC	1N313,2RW03,2ST01,2ST03,2ST06,1N316
4H	2SEQ/RAC	2SQ06.2SQ07,2SQ12,2SQ15,2RA13,2RA17,2RA18
4J	3TBL	3TH02,3TH04
4L	3TBL	3TH03-3TH06
4M	1NUC	1N313
4N	1NUC	1N004,1N013,1N016,1N017,1N020,1N025,1N028 1N034,1N037,1N040,1N043,1N047,1N052
4P	1SEQ/RAC	1SQ09,1SQ10,1SQ21,1ST06,1ST07,1ST10-1ST12,2SQ11
4S	2NUC	2N056,2N057
4T	1NUC	1N313,1N316

4U	1NUC	1N077-1N080,1N105-1N108,1N313,1N316,1N132-1N134 1N169-1N171,2ST01,2ST02,2ST06
4V	2NUC	2N059
4W	2NUC	2N059
4X	2SEQ/RAC	2RA13-2RA16,2SQ06,2SQ07
5A	2NUC	2N026-2N032,2N034-2N036
5B	2NUC	2N045
5C	2NUC	2N046
5D	2NUC	2N033
5E	1NUC	1N066,1N067,1N070-1N073,1N094,1N095,1N098-1N101,1N121, 1N124,1N125,1N127,1N130,1N131,1N151,1N154,1N155,1N157, 1N160,1N161,1N163,1N166,1N167,2N001,2N003,2N005,2N007, 2N009,2N011,2N031,2N034,2N035
5F	1NUC	1N068-1N073,1N080,1N096-1N101,1N108,1N122-1N125,1N128-1N131,1N152-1N155,1N158-1N161,1N164-1N167,2N002,2N003, 2N006,2N008,2N009,2N015,2N032,2N034,2N037,2ST05
5G	2NUC	2N001,2N003,2N007-2N009
5H	2NUC	2N004-2N006,2N010,2N011,2N013,2N015
5K	2SEQ	2SQ06,2SQ09
5L	1SEQ/RAC	1SQ23,1SQ28
5M	2SEQ/RAC	2RA16,2SQ08

5P	2NUC	2N024-2N032,2N034,2N035
5R	1NUC	1N162-1N167,1N171
5S	2NUC	2N038
5T	2NUC	2N038
5U	2NUC	2N013,2N016
5V	2SEQ	2SQ10
5W	2NUC	2N020,2N022
5X	2SRT	2ST04,2ST06
5Y	2SRT	2ST05,2ST06
5Z	3TBL	3TH01,3TH03
6A	1RAC	1RA02,1RA03,1RA06,1RA07,2RA02,2RA03,2RA06,2RA07,2RA10 2RA11,2RA14,2RA15
6B	2SEQ/RAC	2SQ12,2RA17
6C	2RPW	2RW04,2RW05
6D	1SEQ	1SQ21,2SQ11
6E	1SEQ	1SQ21,2SQ11
6F	2NUC	2N050

6H	2NUC	2N051,2N052
6J	2SEQ/RAC	2SQ14,2RA10-2RA12,2RA14-2RA16
6K	2SEQ/RAC	2SQ13,2RA09,2RA11,2RA13,2RA15

TABLE 2. DROP CODE ASSIGNMENT BY TEST

TEST ID	DROP CODES	TEST ID	DROP CODES
1N004	4N	1N069	5F
1N013	4N	1N070	5E,5F
1N016	4N	1N071	5E,5F
1N017	4N	1N072	5E,5F
1N020	4N	1N073	5E,5F
1N025	4N	1N077	4U
1N028	4N	1N078	4U
1N034	4N	1N079	4U
1N037	4N	1N080	4U,5F
1N040	4N	1N094	5E
1N043	4N	1N095	5E
1N047	4N	1N096	5F
1N052	4N	1N097	5F
1N066	5E	1N098	5E,5F
1N067	5E	1N099	5E,5F
1N068	5F	1N100	5E,5F

TEST ID	DROP CODES	TEST ID	DROP CODES
1N101	5E,5F	1N133	4U
1N105	4U	1N134	4U
1N106	4U	1N151	5E
1N107	4U	1N152	5F
1N108	4U,5F	1N153	5F
1N121	5E	1N154	5E,5F
1N122	5F	1N155	5E,5F
1N123	5F	1N157	5E
1N124	5E,5F	1N158	5F
1N125	5E,5F	1N159	5F
1N127	5E	1N160	5E,5F
1N128	5F	1N161	5E,5F
1N129	5F	1N162	5R
1N130	5E,5F	1N163	5E,5R
1N131	5E,5F	1N164	5F,5R
1N132	4U	1N165	5F,5R

TEST ID	DROP CODES	TEST ID	DROP CODES
1N166	5E,5F,5R	1SQ12	3N
1N167	5E,5F,5R	1SQ21	4P,6D,6E
1N169	4U	1SQ23	5L
1N170	4U	1SQ28	5L
1N171	4U,5R	1ST06	4P
1N313	4G,4M,4T,4U	1ST07	4P
1N314	3A	1ST09	4U
1N315	3B	1ST10	4P
1N316	4B,4G,4T,4U	1ST11	4P
1RA02	6A	1ST12	4P
1RA03	6A	1TH02	1M
1RA06	6A	1TH03	1M
1RA07	6A	1TH04	1M
1SQ09	4P	2N001	5E,5G
1SQ10	4P	2N002	5F
1SQ11	3N	2N003	5E,5F,5G

TEST ID	DROP CODES	TEST ID	DROP CODES
2N004	5H	2N026	1D,5A,5P
2N005	5E,5H	2N027	1D,5A,5P
2N006	5F,5H	2N028	1D,5A,5P
2N007	5E,5G	2N029	1D,5A,5P
2N008	5F,5G,1N	2N030	1D,5A,5P
2N009	5E,5F,5G	2N031	1D,5A,5E,5P
2N010	5H	2N032	1D,5A,5F,5P
2N011	5E,5H	2N033	5D
2N013	5U	2N034	1D,5A,5E,5F,5P
2N015	5F,5H	2N035	1D,5A,5E,5P
2N016	5U	2N036	5A
2N020	5W	2N037	5F
2N022	1M,5W	2N038	5S,5T
2N024	1D,5P	2N039	1J
2N025	1D,5P	2N040	1J
		2N041	1J

TEST ID	DROP CODES	TEST ID	DROP CODES
2N042	1J	2RA02	4D,6A
2N043	1C,1D	2RA03	4D,6A
2N045	1D,5B	2RA04	4D
2N046	1D,5C	2RA05	4D
2N050	6F	2RA06	4D,6A
2N051	6H	2RA07	4D,6A
2N052	6H	2RA08	4D
2N053	1L	2RA09	6K
2N054	1L	2RA10	6A,6J,6K
2N056	4S	2RA11	6A,6J,6K
2N057	4S	2RA12	6A,6J,6K
2N058	1G,1H,1K	2RA13	4H,4X,6K
2N059	1E,4V,4W	2RA14	4H,4X,6A,6J
2N060	2A	2RA15	4H,4X,6A,6J,6K
2N061	1A,1B,1F	2RA16	4H,4X,5M,6J
2RA01	4D	2RA17	4H,6B

TEST ID	DROP CODES	TEST ID	DROP CODES
2RA18	4H	2SQ15	4H
2RW01	4F	2ST01	4G,4U
2RW02	4F	2ST02	3P,4U
2RW03	4E,4G	2ST03	3R,3S,4G
2RW04	6C	2ST04	3J,5X
2RW05	6C	2ST05	5F,5Y
2SQ05	3C	2ST06	4G,4U,5X,5Y
2SQ06	3J,4D,5K,4H,4X	2TH02	1M
2SQ07	3J,4D,4H,4X	2TH03	1M
2SQ08	3J,4D,5M	2TH04	1M
2SQ09	5K	3TH01	1M,5Z
2SQ10	5V	3TH02	1M,4J
2SQ11	4P,6D,6E	3TH03	1M,4L,5Z
2SQ12	4H,6B	3TH04	1M,4J,4L
2SQ13	6K	3TH05	4L
2SQ14	6J	3TH06	4L

APPENDIX III

INITIATING ENVIRONMENTAL DATA

Environmental Data is entered on the form shown in Figure 3. The number of each entry is related to a statement found in the Environmental Data questionnaire that follows Figure 3. The information required is found in the implementor's COBOL manual.

The method for assigning the indicative computer-name is discussed in 3.1.1.1.2.

Figures 4, 5 and 6 are presented as an aid to understanding the various environmental input entries and how they are treated by the PFM run. Figure 4 shows a form similar to that appearing in Figure 3, filled out with entries for a fictional implementation - the XYZ 8795, model 1. Figure 5 shows the corresponding result of processing this information through PFM. The relationship between the user's entries on Figure 4 and the PFM output of Figure 5 is expressed on Figure 6. This indicates for each output card (ETOnnn), the user's entry from which it came and any transformations applied by PFM. In the absence of an entry in the "Comments" column, the PFM output is taken directly from the user input. When the user input is blank, PFM places an 'N' in column 7 of the corresponding output card.

	INDICATIVE COMPUTER NAME	ACTION				
01	(1) OBJECT-COMPUTER NAME		(2) MEMORY-SIZE CLAUSE ENTRY	(3) USABLE SIZE (CHAR)	(4) PRINT LINE SIZE	
02	(5) SWITCH-STATUS SWITCH		(6) RERUN SWITCH	(7) "FIRST-SWITCH" OR BLANK		STATUS
03	(9) ACCEPT FROM DEVICE-NAME		(10) DISPLAY UPON DEVICE-NAME	(11) 1 SPACE	(12) 2 SPACE	(13) EJECT (14) CODE (15) LITERAL
04	(16) TAPE UNIT 1		(17) TAPE UNIT 2			
05	(18) TAPE UNIT 3		(19) DIRECT ACCESS UNIT 1			
06	(20) DIRECT ACCESS UNIT 2		(21) CARD READER			

Figure 3. Environmental Data Card Format

22	PRINTER											
23	SORT UNIT 1											
24	SORT UNIT 2											
25	SORT UNIT 3											
26	SORT UNIT 4											
27	FILE-LIMIT PICTURE	USAGE	LIMIT 1	LIMIT 2	INCREMENT 1	LIMIT 3	LIMIT 4	INCREMENT 2				
28	ACTUAL-KEY PICTURE	USAGE	ACTUAL-KEY LITERAL 1	ACTUAL-KEY LITERAL 2	ACTUAL-KEY LITERAL 3	ACTUAL-KEY LITERAL 4	INCREMENT 1	INCREMENT 2	IMPLEMENTOR-NAME FOR RERUN			
29	VALUE OF - IMPLEMENTOR-NAME 1						VALUE OF LITERAL 1		VALUE OF - IMPLEMENTOR-NAME 2			
30	VALUE OF LITERAL 2		VALUE OF d-n PICTURE									

Figure 3. Environmental Data Card Format (Cont'd)

FIELD NUMBER VS INFORMATION
ON
ENVIRONMENTAL DATA CARDS

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

FIELD NO.	OBJECT COMPUTER	MAXIMUM NO. OF CHARACTERS IN ENTRY
1.	The implementor's name for OBJECT-COMPUTER.	30
2.	The memory size and unit of measure as specified for the MEMORY SIZE clause.	30
3.	The memory size of the object computer in characters/bytes, written as a six digit integer.	6
4.	The print line size of the on-line printer written as a three digit integer less than or equal to 120. 120 assumed if omitted.	3
5.	Name specified by implementor for a switch testable by the switch-status test.	30
6.	Name specified by implementor for the switch whose setting indicates a RERUN is to be taken (use only if 48=Y)	30
7.	If the "mnemonic-name" option is available for switch-names enter FIRST-SWITCH.	12
8.	If both the ON STATUS and OFF STATUS options are available enter a '1'. If only ON STATUS is available, enter '2'.	1
	OTHER-NAMES	
9.	Name specified for the device available using the FROM option of ACCEPT.	30
10.	Name specified for the device available using the UPON option of DISPLAY.	30
11.	If the mnemonic-name option of the ADVANCING feature of WRITE has been implemented, write the character designated by the implementor to denote single space, enclosed in quotes if necessary.	3
12.	Write the charactor denoting double space.	3
13.	Write tho charactor denoting page eject.	3
14.	If tho "CODE mnemonic-name" option of tho RD clause is implemented enter a "Y".	1
INPUT-OUTPUT SECTION		
	FILE-CONTROL	
15.	If the 'integer' option of the ASSIGN clause is available, enter a 'Y'.	1

16.	Entries 16 through 22 are concerned with the implementor-names for input-output devices. Of these, the entries for the 3 tape units, the card reader and the printer are mandatory (the card reader and/or printer may be system-units or any sequential device not previously named). Enter the implementor-name for the first tape-unit.	30
17.	Enter the implementor-name for the second tape unit.	30
18.	Enter the implementor-name for the third tape unit.	30
19.	Enter the implementor-name for the first of two mass-storage devices (may be omitted if no device available).	30
20.	Enter the implementor-name of the second mass-storage device. (May be omitted if no device available.)	30
21.	Enter the implementor-name of the card reader.	30
22.	Enter the implementor-name of the printer.	
23,24. 25 & 26.	Enter the implementor-name of the Sort units 1 through 4 if the SORT feature has been implemented.	30
27.	For the data-name option of the FILE-LIMIT clause, enter the description (picture and usage) necessary to describe data-name in entries 27 and 28. Enter the PICTURE (e.g., 9(5)).	6
28.	Enter the USAGE of data-name: C for COMPUTATIONAL or D for DISPLAY.	1
29.	Enter the literal designated by the implementor as meaning the lowest available mass-storage address for the FILE-LIMIT clause.	6
30.	Enter the literal that indicates a mass storage address twenty records of 100 characters each higher than the address specified in 29.	6
31.	If a second set of FILE-LIMITS can be specified in the implementation enter a number that when added to the literals specified in 29 and 30 specifies a second available area of this mass-storage file.	6
32,33 & 34	If a second mass-storage device has been specified (see entry 20), specify the lower and upper limits, and increment for it in the same way as specified under 29, 30 and 31.	6,6,6
35.	Enter the PICTURE required in the description of the data-name used in the ACTUAL KEY clause (e.g., 9(5)).	6
36.	Enter the USAGE required in the description of the data-name used in the ACTUAL KEY clause: C for COMPUTATIONAL or D for DISPLAY.	1
37.	Enter the literal value that when placed in the data-name of ACTUAL KEY will indicate the lowest mass-storage address associated with the first area allocated for the first mass-storage device (see 29 and 30).	6

38.	Enter the literal value that when placed in the data-name of ACTUAL KEY will indicate the lowest mass-storage address associated with the second area allocated for the first mass-storage device (see 31).	6
39.	Enter the literal value that when placed in the data-name of ACTUAL KEY will indicate the lowest mass-storage address associated with the first area allocated for the second mass-storage device.	6
40.	Enter the literal value that when placed in the data-name of ACTUAL KEY will indicate the lowest mass-storage address associated with the second area allocated for the second mass-storage device (see 34).	6
41.	Enter the literal value that corresponds to an increment of one 100 character logical record to the ACTUAL KEY of the first mass-storage device.	6
42.	Enter the literal value that corresponds to an increment of one 100 character logical record to the ACTUAL KEY of the second mass-storage device.	6
	I-O-CONTROL	
43.	If an implementor-name is available to the RERUN clause, enter it here.	30
	Items 44-48 deal with the versions of the RERUN clause available. Enter a 'Y' for each option available: each option so marked will be tested.	
44.	END OF REEL	1
45.	END OF UNIT	1
46.	integer RECORDS (item 43 above must be filled in)	1
47.	integer CLOCK-UNITS (item 43 above must be filled in)	1
48.	condition-name (switch-name must be entered in 6)	1
	DATA DIVISION FILE SECTION	
49.	Enter the implementor's fixed-name for the file identification to be used in the VALUE OF clause.	30
50.	Enter any literal value that corresponds to the implementor's rules for the content of the file identification. Include quotes if required.	15
51.	Enter the implementor's fixed name for an item in the standard label record (other than file identification) that is inserted into output labels (e.g., Retention-period).	30

52. Enter the literal corresponding to the implementor-name entered in 51. 15

53. If the data-name option of the VALUE OF clause is available, enter the PICTURE required by the implementor's file identification. 6

XYZ0879501A

01	XYZ-8795	32768 WØRDS	196 608	120
02	SWITCH1	SWITCH2	FIRST-SWITCH1	
03	CØNSØLE	CØNSØLE	'1' '2' 'E' YY	
04	TAPE1	TAPE2		
05	TAPE3	DISK1		
06	DISK2	READER		

Figure 4. Environmental Data Cards for XYZ-8795

01	PRINTER				SØRT1						
03	SØRT2				SØRT3						
00	SØRT4				59(8)	C000001	000021	000030	000001	000021	000030
40	5(98)	C000001	000021	000001	000021	000001	000001	RERUN-NAME			
44	ID				'USASI'			RETENTIØN-PERIØD			
42	002	9(6)									

Figure 4. Environmental Data Cards For XYZ-8795 (Cont'd)

```
F XYZ0879501A
 -------------------------      ENV ADD
F XYZ0879501Y
FT0010                    XYZ-8795
FT0020                    XYZ-8795
FT0030      MEMORY SIZE   32768 WORDS.
FT0040                    019660
FT050                                   PICTURE X(121).
FT060          120
FT0070      SWITCH1
FT0080              IS    FIRST-SWITCH
FT0090      SWITCH2
FT0100         ON  STATUS IS
FT0110         OFF STATUS IS
FT0120      CONSOLE
FT0130      CONSOLE
FT0140      :1:
FT0150      :2:
FT0160      :F:
FT0170      :A:
FT0180      1
FT0190                    TAPE1
FT0200                    TAPE2
FT0210                    TAPE3
FT0220                    DISK1
FT0230                    DISK2
FT0240                    READER
FT0250                    PRINTER
FT0260                    SORT1
FT0270                    SORT2
FT0280                    SORT3
FT0290                    SORT4
FT0300         PICTURE    S9(8)        COMPUTATIONAL.
FT0310                    000001
FT0320                    000021
FT0330                    000030
FT0340                    000001
FT0350                    000021
FT0360                    000030
FT0370         PICTURE    S9(8)        COMPUTATIONAL.
FT0380                    000001
FT0390                    000021
FT0400                    000001
FT0410                    000021
FT0420                    000001
FT0430                    000001
FT0440      RERUN ON       RERUN-NAME
FT0450         EVERY END OF REEL
FT0460N        EVERY END OF UNIT
FT0470N        EVERY 10 RECORDS
FT0480N        EVERY 1 CLOCK-UNITS
FT0490N        EVERY
FT0500      VALUE OF ID                                  IS :USASI:
FT0510      VALUE OF RETENTION-PERIOD                    IS 002
FT0520         PICTURE   9(6)
```

Figure 5. Output Listing of Environmental Data - XYZ-8795

OUTPUT CARD NO.	FROM ENTRY	COMMENT
ET0010	1	
ET0020	1	
ET0030	2	
ET0040	3	Entry 3 is divided by 10 and zero-filled to 6 digits.
ET0050	4	Entry 4 plus 1.
ET0060	4	
ET0070	5	
ET0080	7	
ET0090	6	
ET0100	8	If entry 8 = 1 or 2, column 7 is blank, otherwise N.
ET0110	8	If entry 8 = 1, column 7 is blank, otherwise N.
ET0120	9	
ET0130	10	
ET0140	11	
ET0150	12	
ET0160	13	
ET0170	14	If 14 = Y, column 7 is blank, otherwise N.
ET0180	15	If 14 = Y, column 7 is blank, otherwise N.
ET0190	16	
ET0200	17	
ET0210	18	
ET0220	19	
ET0230	20	
ET0240	21	
ET0250	22	
ET0260	23	
ET0270	24	
ET0280	25	
ET0290	26	

Figure 6. (1)
Relationship Between User Input
and PFM Environmental Output

OUTPUT CARD NO.	FROM ENTRY	COMMENT
ET0300	27, 28	Entry 28 is first letter of Usage.
ET0310	29	
ET0320	30	
ET0330	31	
ET0340	32	
ET0350	33	
ET0360	34	
ET0370	35, 36	Entry 36 is first letter of Usage.
ET0380	37	
ET0390	38	
ET0400	39	
ET0410	40	
ET0420	41	
ET0430	42	
ET0440	43	
ET0450	44	If entry is a Y, column 7 is blank, otherwise N.
ET0460	45	If entry is a Y, column 7 is blank, otherwise N.
ET0470	46	If entry is a Y, column 7 is blank, otherwise N.
ET0480	47	If entry is a Y, column 7 is blank, otherwise N.
ET0490	48	If entry is a Y, column 7 is blank, otherwise N.
ET0500	49, 50	
ET0510	51, 52	
ET0520	53	

Figure 6. (2)
Relationship Between User Input
and PFM Environmental Output

APPENDIX IV

TEST DIRECTORY

NUC	1	90
	2	108
TBL	1	114
	2	115
	3	116
SEQ	1	117
	2	119
RAC	1	120
	2	121
SRT	1	123
	2	125
RPW	1	126
	2	127
SEG	1	128
	2	130
LIB	1	131
	2	132

Test ID	Summary of Test	Breakdown of Printed Results
1N001	MOVE GRP-GROUP-MOVE-FROM TO GRP-GROUP-MOVE-TO.	Receiving field is printed.
1N002	MOVE GRP-ALPHABETIC TO WRK-AN-00026.	Receiving field is printed.
1N003	MOVE GRP-ALPHANUMERIC TO WRK-XN-00049.	Receiving field is printed.
1N004	MOVE GRP-ALPHANUMERIC TO AE-0001.	Receiving field is printed.
1N005	MOVE GRP-NUMERIC TO WRK-DU-10V00.	Receiving field is printed.
1N006	MOVE GRP-NUMERIC TO NE-0001.	Receiving field is printed.
1N007	MOVE ALPHABET-AN-00026 TO GRP-WRK-AN-00026.	Receiving field is printed.
1N008	MOVE ALPHABET-AN-00026 TO WRK-AN-00026.	Receiving field is printed.
1N009	MOVE ALPHABET-AN-00026 TO WRK-XN-00049.	Receiving field is printed.
1N010	MOVE ALPHANUMERIC-XN-00049 TO GRP-WRK-XN-00049.	Receiving field is printed.
1N011	MOVE ALPHANUMERIC-XN-00049 TO WRK-AN-00026.	Receiving field is printed.
1N012	MOVE ALPHANUMERIC-XN-00049 TO WRK-XN-00049.	Receiving field is printed.
1N013	MOVE ALPHANUMERIC-XN-00049 TO AE-0001.	Receiving field is printed.
1N014	MOVE ALPHANUMERIC-XN-00049 TO WRK-DU-10V00.	Receiving field is printed.
1N015	MOVE ALPHANUMERIC-XN-00049 TO NE-0002.	Receiving field is printed.
1N016	MOVE AE-0001 TO SUP-WK-A.	Receiving field is printed.
1N017	MOVE AE-0001 TO AE-0002.	Receiving field is printed.
1N018	MOVE DIGITS-DU-10V00 TO GRP-WRK-DU-10V00.	Receiving field is printed.
1N019	MOVE DIGITS-DU-10V00 TO WRK-XN-00049.	Receiving field is printed.
1N020	MOVE DIGITS-DU-10V00 TO AE-0002.	Receiving field is printed.
1N021	MOVE DIGITS-DU-10V00 TO WRK-DU-10V00.	Receiving field is printed.
1N022	MOVE DIGITS-DU-06V04-S TO NE-0001.	Receiving field is printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N023	MOVE NE-0001 TO GRP-WRK-XN-00049.	Receiving field is printed.
1N024	MOVE NE-0001 TO WRK-XN-00049.	Receiving field is printed.
1N025	MOVE NE-0001 TO AE-0002.	Receiving field is printed.
1N026	MOVE ZERO TO GRP-WRK-DU-10V00.	Receiving field is printed.
1N027	MOVE ZERO TO WRK-XN-00049.	Receiving field is printed.
1N028	MOVE ZERO TO AE-0002.	Receiving field is printed.
1N029	MOVE ZERO TO WRK-DU-10V00.	Receiving field is printed.
1N030	MOVE ZERO TO NE-0001.	Receiving field is printed.
1N031	MOVE SPACE TO GRP-WRK-DU-10V00.	Receiving field is printed.
1N032	MOVE SPACE TO WRK-AN-00026.	Receiving field is printed.
1N033	MOVE SPACE TO WRK-XN-00049.	Receiving field is printed.
1N034	MOVE SPACE TO AE-0002.	Receiving field is printed.
1N035	MOVE HIGH-VALUE TO GRP-WRK-DU-10V00.	Receiving field is printed.
1N036	MOVE HIGH-VALUE TO WRK-XN-00049.	Receiving field is printed.
1N037	MOVE HIGH-VALUE TO AE-0002.	Receiving field is printed.
1N038	MOVE LOW-VALUE TO GRP-WRK-DU-10V00.	Receiving field is printed.
1N039	MOVE LOW-VALUE TO WRK-XN-00049.	Receiving field is printed.
1N040	MOVE LOW-VALUE TO AE-0002.	Receiving field is printed.
1N041	MOVE QUOTE TO GRP-WRK-DU-10V00.	Receiving field is printed.
1N042	MOVE QUOTE TO WRK-XN-00049.	Receiving field is printed.
1N043	MOVE QUOTE TO AE-0002.	Receiving field is printed.
1N044	MOVE "A1B2C3D4E5" TO GRP-WRK-DU-10V00.	Receiving field is printed.
1N045	MOVE "ABCDEFGHIJK" TO WRK-AN-00026.	Receiving field is printed.
1N046	MOVE "1A2B3C4D5E6F" TO WRK-XN-00049.	Receiving field is printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N047	MOVE "1Z2Y3X4W5V" TO AE-0002.	Receiving field is printed.
1N048	MOVE "9876543210" TO WRK-DU-10V00.	Receiving field is printed.
1N049	MOVE "9876543210" TO NE-0002.	Receiving field is printed.
1N050	MOVE 0123456789 TO GRP-WRK-DU-10V00.	Receiving field is printed.
1N051	MOVE 0918273645 TO WRK-XN-00049.	Receiving field is printed.
1N052	MOVE 019823 TO AE-0002.	Receiving field is printed.
1N053	MOVE 9876543210 TO WRK-DU-10V00.	Receiving field is printed.
1N054	MOVE 00012345 TO NE-0002.	Receiving field is printed.
1N055	MOVE 000011.1223 TO NE-0001.	Receiving field is printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N061	ADD A18ONES-DS-18V00 TO WRK-DS-18V00.	Arithmetic result is printed.
1N062	ADD A10ONES-DS-10V00 A05ONES-DS-05V00 TO WRK-DS-10V00.	Arithmetic result is printed.
1N063	ADD A02ONES-DS-02V00 A10ONES-DS-10V00 A05ONES-DS-05V00 TO WRK-DS-10V00.	Arithmetic result is printed.
1N064	ADD A06THREES-DS-03V03 A12THREES-DS-06V06 GIVING WRK-DS-09V09.	Arithmetic result is printed.
1N065	ADD A05ONES-DS-05V00 A05ONES-DS-00V05 A12THREES-DS-06V06 A06THREES-DS-03V03 GIVING WRK-DS-06V06.	Arithmetic result is printed.
1N066	ADD 55554.5 TO WRK-DS-05V00 ROUNDED.	Arithmetic result is printed.
1N067	ADD A05ONES-DS-00V05 A12THREES-DS-06V06 A05ONES-DS-00V05 GIVING WRK-DS-05V00 ROUNDED.	Arithmetic result is printed.
1N068	ADD -99 TO WRK-DS-02V00 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N069	ADD A12ONES-DS-12V00 ZERO GIVING WRK-DS-10V00 ON SIZE ERROR MOVE "1" TO WRK-XN-0001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N070	ADD 33333 A06THREES-DS-03V03 A12THREES-DS-06V06 TO WRK-DS-05V00 ROUNDED ON SIZE ERROR.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N071	ADD A12THREES-DS-06V06 333333 A06THREES-DS-03V03 TO WRK-DS-06V06 ROUNDED ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).

Test ID	Summary of Test	Breakdown of Printed Results
1N072	ADD 33333 A06THREES-DS-03V03 A12THREES-DS-06V06 GIVING WRK-DS-05V00 ROUNDED ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N073	ADD A12THREES-DS-06V06 333333 A06THREES--DS-03V03 GIVING WRK-DS-06V06 ROUNDED ON SIZE ERROR.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N074	ADD A99-DS-02V00 A03ONES-DS-02V01 A06ONES-DS-03V03 A08TWOS-DS-02V06 -1.1111111 +.11111111 A01ONE-DS-P0801 GIVING WRK-DS-03V10.	Arithmetic result is printed.
1N075	ADD A01ONE-DS-P0801 +.11111111 -1.1111111 A08TWOS-DS-02V06 A06ONES-DS 03V03 A03ONES-DS-02V01 A99-DS-02V00 GIVING WRK-DS-03V10.	Arithmetic result is printed.
1N076	ADD A08TWOS-DS-02V06 A99-DS-02V00 -1.1111111 A03ONES-DS-02V01 A01ONE-DS P0801 +.11111111 A06ONES-DS-03V03 GIVING WRK-DS-03V10.	Arithmetic result is printed.
1N077	ADD A18ONES-DS-18V00 TO WRK-CS-18V00.	Arithmetic result is printed.
1N078	ADD A18CNES-CS-18V00 TO WRK-DS-18V00.	Arithmetic result is printed.
1N079	ADD A99-CS-02V00 TO WRK-CS-02V02.	Arithmetic result is printed.
1N080	ADD A99-CS-02V00 TO WRK-CS-02V02 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).

Test ID	Summary of Test	Breakdown of Printed Results
1N090	SUBTRACT A18ONES-DS-18V00 FROM WRK-DS-18V00.	Arithmetic result is printed.
1N091	SUBTRACT A05ONES-DS-05V00 A05ONES-DS-00V05 A06ONES-DS-03V03 FROM WRK-DS-06V06.	Arithmetic result is printed.
1N092	SUBTRACT A06THREES-DS-03V03 FROM A12THREES-DS-06V06 GIVING WRK-DS-06V06.	Arithmetic result is printed.
1N093	SUBTRACT A05ONES-DS-05V00 A05ONES-DS-00V05 A12THREES-DS-06V06 A06THREES-DS-03V03 FROM ZERO GIVING WRK-DS-06V06.	Arithmetic result is printed.
1N094	SUBTRACT A99-DS-02V00 FROM WRK-DS-0201P ROUNDED.	Arithmetic result is printed.
1N095	SUBTRACT A05ONES-DS-05V00 -11111 AZERO-DS-05V05 FROM WRK-DS-06V06 GIVING WRK-DS-06V00 ROUNDED.	Arithmetic result is printed.
1N096	SUBTRACT A99-DS-02V00 FROM WRK-DS-02V00 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N097	SUBTRACT A12ONES-DS-12V00 FROM ZERO GIVING WRK-DS-10V00 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N098	SUBTRACT 33333 A06THREES-DS-03V03 A12THREES-DS-06V06 FROM WRK-DS-05V00 ROUNDED ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N099	SUBTRACT A12THREES-DS-06V06 333333 A06THREES-DS-03V03 FROM WRK-DS-06V06 ROUNDED ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).

Test ID	Summary of Test	Breakdown of Printed Results
1N100	SUBTRACT 33333 A06THREES-DS-03V03 A12THREES-DS-06V06 FROM -1000000 WRK-DS-05V00 ROUNDED ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N101	SUBTRACT A12THREES-DS-06V06 333333 A06THREES-DS-03V03 -.0000009 FROM -.1000000 GIVING WRK-DS-06V06 ROUNDED ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N102	SUBTRACT A99-DS-02V00 A03NES-DS-02V01 A06ONES-DS-03V03 A08TWOS-DS-02V06 -1.1111111 +.11111111 A01ONE-DS-P0801 FROM -1000.000000 GIVING WRK-DS-03V10.	Arithmetic result is printed.
1N103	SUBTRACT A01ONE-DS-P0801 +.11111111 -1.1111111 A08TWOS-DS-02V06 A06ONES-DS 03V03 A03ONES-DS-02V01 A99-DS-02V00 FROM -1000.000000 GIVING WRK-DS-03V10.	Arithmetic result is printed.
1N104	SUBTRACT A08TWOS-DS-02V06 A99-DS-02V00 -1.1111111 A03ONES-DS-02V01 A01ONE-DS-P0801 +.11111111 A06ONES-DS-03V03 FROM -1000.000000 GIVING WRK-DS-03V10.	Arithmetic result is printed.
1N105	SUBTRACT A18ONES-DS-18V00 FROM WRK-CS-18V00.	Arithmetic result is printed.
1N106	SUBTRACT A18ONES-CS-18V00 FROM WRK-DS-18V00.	Arithmetic result is printed.
1N107	SUBTRACT A99-CS-02V00 FROM WRK-CS-02V02.	Arithmetic result is printed.
1N108	SUBTRACT -99 FROM WRK-CS-02V02 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).

Test ID	Summary of Test	Breakdown of Printed Results
1N120	MULTIPLY A06THREES-DS-03V03 BY WRK-DS-18V00.	Arithmetic result is printed.
1N121	MULTIPLY 0.4 BY WRK-DS-06V06 ROUNDED.	Arithmetic result is printed.
1N122	MULTIPLY A12THREES-DS-06V06 BY WRK-DS-10V00 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N123	MULTIPLY AZERO-DS-05V05 ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N124	MULTIPLY 99.5 BY WRK-DS-02V00 ROUNDED ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N125	MULTIPLY 99.4 BY WRK-DS-02V00 ROUNDED ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N126	MULTIPLY A06THREES-DS-03V03 BY A12THREES-DS-06V06 GIVING WRK-DS-09V09.	Arithmetic result is printed.
1N127	MULTIPLY A06THREES-DS-03V03 BY A06THREES-DS-03V03 GIVING WRK-DS-10V00 ROUNDED.	Arithmetic result is printed.
1N128	MULTIPLY A05ONES-DS-10V00 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N129	MULTIPLY A01ONES-DS-P0801 BY A12ONES-DS-12V00 GIVING WRK-DS-10V00 ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).

Test ID	Summary of Test	Breakdown of Printed Results
1N130	MULTIPLY 1.5 BY A100NES-DS-10V00 GIVING WRK-DS-10V00 ROUNDED ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N131	MULTIPLY A010NE-DS-P0801 BY A180NES-DS-18V00 GIVING WRK-DS-09V08 ROUNDED ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N132	MULTIPLY A010NE-CS-00V01 BY WRK-DS-0201P.	Arithmetic results are printed.
1N133	MULTIPLY A010NE-DS-P0801 BY WRK-CS-18V00.	Arithmetic results are printed.
1N134	MULTIPLY A99-CS-02V00 BY A010NE-CS-00V01 GIVING WRK-CS-02V02.	Arithmetic results are printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N150	DIVIDE A99-DS-02V00 INTO WRK-DS-18V00.	Arithmetic result is printed.
1N151	DIVIDE 4 INTO WRK-DS-12V00 ROUNDED.	Arithmetic result is printed.
1N152	DIVIDE 0.1 INTO WRK-DS-01V00 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N153	DIVIDE A01ONE-DS-P0801 INTO WRK-DS-09V00 ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N154	DIVIDE AZERO-DS-05V05 INTO WRK-DS01V00 ROUNDED ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N155	DIVIDE AONES-DS-09V09 INTO WRK-DS-09V09 ROUNDED ON SIZE ERROR MOVE "0" TO WRK-XN 00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N156	DIVIDE -10.9 INTO A02TWOS-DS-02V00 GIVING WRK-DS-01V00.	Arithmetic result is printed.
1N157	DIVIDE WRK-DS-03V10 INTO A01ONE-DS-P0801 GIVING WRK-DS-18V00 ROUNDED.	Arithmetic result is printed.
1N158	DIVIDE AZERO-DS-05V05 INTO A99-DS-02V00 GIVING WRK-DS-18V00 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N159	DIVIDE AONES-DS-09V09 INTO WRK-DS-09V09 GIVING WRK-DS-09V09 ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).

Test ID	Summary of Test	Breakdown of Printed Results
1N160	DIVIDE WRK-DS-09V09 INTO A050NES-DS-00V05 GIVING WRK-DS-0201P ROUNDED ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N161	DIVIDE A02TW0S-DS-02V00 INTO A02TW0S-DS-03V02 GIVING WRK-DS-01V00 ROUNDED ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N162	DIVIDE A02TW0S-DS-02V00 BY -10.9 GIVING WRK-DS-01V00.	Arithmetic result is printed.
1N163	DIVIDE A010NE-DS-P0801 BY WRK-DS- 03V10 GIVING WRK-DS-18V00 ROUNDED.	Arithmetic result is printed.
1N164	DIVIDE A99-DS-02V00 BY AZERO-DS-05V05 GIVING WRK-DS-18V00 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N165	DIVIDE WRK-DS-09V09 BY A0NES-DS-09V09 GIVING WRK-DS-09V09 ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N166	DIVIDE A050NES-DS-00V05 BY WRK-DS-09V09 GIVING WRK-DS-0201P ROUNDED ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N167	DIVIDE A02TW0S-DS-03V02 BY A02TW0S-DS 02V00 GIVING WRK-DS-01V00 ROUNDED ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
1N168	DIVIDE A99-DS-02V00 INTO WRK-DS-02V00.	Arithmetic result is printed.
1N169	DIVIDE A990-DS-0201P INTO WRK-CS-02V02.	Arithmetic result is printed.
1N170	DIVIDE A010NE-CS-00V01 INTO A99-CS-02V00 GIVING WRK-DS-05V00.	Arithmetic result is printed.
1N171	DIVIDE A99-CS-02V00 BY A010NE-CS-00V01 GIVING WRK-DS-05V00.	Arithmetic result is printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N200	PERFORM paragraph-name. PERFORM section-name.	Execution sequence indicators are printed.
1N201	PERFORM paragraph-name integer TIMES. PERFORM section-name data-name TIMES.	Execution sequence indicators are printed.
1N202	PERFORM paragraph-name THRU paragraph-name. PERFORM section-name THRU section-name. PERFORM paragraph-name THRU section-name.	Execution sequence indicators are printed.
1N203	PERFORM paragraph-name THRU paragraph-name integer TIMES. PERFORM section-name THRU section-name integer TIMES. PERFORM section-name THRU paragraph-name integer TIMES. PERFORM paragraph-name THRU section-name integer TIMES.	Execution sequence indicators are printed.
1N210	Tests nested PERFORM statements.	Execution sequence indicators are printed.
1N215	Tests the use of EXIT.	Execution sequence indicators are printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N230	EXAMINE NDATA-DS-09V09 TALLYING UNTIL FIRST 9.	The value of TALLY is printed.
1N231	EXAMINE XDATA-XN-00018 TALLYING ALL "0".	The value of TALLY is printed.
1N232	EXAMINE XDATA-XN-00018 TALLYING LEADING SPACE.	The value of TALLY is printed.
1N237	EXAMINE WRK-DS-09V09 TALLYING UNTIL FIRST ZERO REPLACING BY 9.	The value of TALLY and the examined item are printed.
1N238	EXAMINE WRK-XN-00018 TALLYING ALL ZERO REPLACING BY SPACE.	The value of TALLY and the examined item are printed.
1N239	EXAMINE WRK-XN-00018 TALLYING LEADING "0" REPLACING BY "9".	The value of TALLY and the examined item are printed.
1N242	EXAMINE WRK-DS-09V09 REPLACING ALL 0 BY 9.	The value of the examined item is printed.
1N243	EXAMINE WRK-XN-00018 REPLACING LEADING "0" BY "Z".	The value of the examined item is printed.
1N244	EXAMINE WRK-XN-00018 REPLACING UNTIL SPACE BY ZERO.	The value of the examined item is printed.
1N245	EXAMINE WRK-DS-09V09 REPLACING FIRST 8 BY ZERO.	The value of the examined item is printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N250	IF AZERO-DS-05V05 IS EQUAL TO ZERO MOVE "1" TO SUP-WK-B (1).	The result of the comparison is printed.
1N251	IF SPACE IS EQUAL TO SUP-WK-A MOVE "0" TO SUP-WK-B (1).	The result of the comparison is printed.
1N252	IF A18ONES-DS-18V00 IS EQUAL TO ONES-XN-00018 MOVE "1" TO SUP-WK-B (1).	The result of the comparison is printed.
1N253	IF TWOS-XN-00002 IS EQUAL TO A99-DS-02V00 MOVE "0" TO SUP-WK-B (1).	The result of the comparison is printed.
1N254	IF A99-DS-02V00 IS LESS THAN A18ONES-DS-09V09 MOVE "1" TO SUP-WK-B (1) ELSE MOVE "0" TO SUP-WK-B (2).	The result of the comparison is printed.
1N255	IF "11" IS LESS THAN ONES-XN-00002 MOVE "0" TP SUP-WK-B (1) ELSE MOVE "1" TO SUP-WK-B (2).	The result of the comparison is printed.
1N256	IF A02TWOS-DU-02V00 IS LESS THAN ONES-XN-00002 MOVE "1" TO SUP-WK-B (1) ELSE MOVE "0" TO SUP-WK-B (2).	The result of the comparison is printed.
1N257	IF TWOS-XN-00002 IS LESS THAN A02TWOS-DU-02V00 MOVE "0" TO SUP-WK-B (1) ELSE MOVE "1" TO SUP-WK-B (2).	The result of the comparison is printed.
1N258	IF A99-DS-02V00 IS GREATER THAN 88.9 NEXT SENTENCE ELSE MOVE "0" TO SUP-WK-B (1).	The result of the comparison is printed.
1N259	IF ONES-XN-00002 IS GREATER THAN TWOS-XN-00002 NEXT SENTENCE ELSE MOVE "1" TO SUP-WK-B (1) GO TO TEST-1NUC-259-A.	The result of the comparison is printed.
1N260	IF A02TWOS-DU-02V00 IS GREATER THAN ONES-XN-00002 NEXT SENTENCE ELSE MOVE "0" TO SUP-WK-B (1).	The result of the comparison is printed.
1N261	IF TWOS-XN-00002 IS GREATER THAN A02TWOS-DU-02V00 NEXT SENTENCE ELSE MOVE "1" TO SUP-WK-B (1) GO TO TEST-INUC-261-A.	The result of the comparison is printed.
1N262	IF ZERO IS NOT EQUAL TO SUP-WK-A MOVE "1" TO SUP-WK-B (1) GO TO TEST-1NUC-262-A ELSE NEXT SENTENCE.	The result of the comparison is printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N263	IF A02TWOS-DU-02V00 IS NOT EQUAL TO A02TWOS-DS-03V02 MOVE "0" TO SUP-WK-B (1) GO TO TEST-1NUC-263-A ELSE NEXT SENTENCE.	The result of the comparison is printed.
1N264	IF TWOS-XN-00002 IS NOT LESS THAN ONES-XN-00002 MOVE "1" TO SUP-WK-B (1) GO TO TEST-1NUC-264-A ELSE NEXT SENTENCE.	The result of the comparison is printed.
1N265	IF 0.0000000001 IS NOT LESS THAN A01ONES-DS-P0801 MOVE "0" TO SUP-WK-B (1) GO TO TEST-1NUC-265-A ELSE NEXT SENTENCE.	The result of the comparison is printed.
1N266	IF ONES-XN-00002 IS NOT GREATER THAN TWOS-XN-00002 MOVE "1" TO SUP-WK-B (1) GO TO TEST-1NUC-266-A ELSE NEXT SENTENCE.	The result of the comparison is printed.
1N267	IF A990-DS-0201P IS NOT GREATER THAN A99-DS-02V00MOVE "0" TO SUP-WK-B (1) GO TO TEST-1NUC-267-A ELSE NEXT SENTENCE.	The result of the comparison is printed.
1N270	IF ONES-XN-00018 IS NUMERIC MOVE "1" TO SUP-WK-B (1).	The result of the test is printed.
1N271	IF A02TWOS-DS-03V02 IS NUMERIC MOVE "1" TO SUP-WK-B (1).	The result of the test is printed.
1N272	IF XDATA-XN-00018 IS NUMERIC MOVE "0" TO SUP-WK-B (1) GO TO TEST-1NUC-272-A.	The result of the test is printed.
1N273	IF XDATA-DS-18V00-S IS NUMERIC MOVE "0" TO SUP-WK-B (1) GO TO TEST-1NUC-273-A.	The result of the test is printed.
1N274	IF SUP-WK-A IS NOT NUMERIC MOVE "1" TO SUP-WK-B (1) GO TO TEST-1NUC-274-A.	The result of the test is printed.
1N275	IF XDATA-DS-18V00-S IS NOT NUMERIC MOVE "1" TO SUP-WK-B (1).	The result of the test is printed.
1N276	IF SUP-WK-A IS NOT NUMERIC MOVE "0" TO SUP-WK-B (1) GO TO TEST-1NUC-277-A.	The result of the test is printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N277	IF A990-DS-0201P IS NOT NUMERIC MOVE "0" TO SUP-WK-B (1) GO TO TEST-1NUC-277-A.	The result of the test is printed.
1N278	IF YADATA-XN-00010 IS ALPHABETIC MOVE "1" TO SUP-WK-B (1).	The result of the test is printed.
1N279	IF SUP-WK-A IS ALPHABETIC MOVE "X" TO SUP-WK-B (1) GO TO TEST-1NUC-279-A.	The result of the test is printed.
1N280	IF XDATA-XN-00018 IS NOT ALPHABETIC MOVE "1" TO SUP-WK-B (1).	The result of the test is printed.
1N281	IF YADATA-XN-00010 IS NOT ALPHABETIC MOVE "0" TO SUP-WK-B (1) GO TO TEST-1NUC-281-A.	The result of the test is printed.
1N282	IF IFNUM-DU-01V00 IS NUMERIC MOVE "0" TO SUP-WK-B (1) GO TO TEST-1NUC-282-A.	The result of the test is printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N300	Tests format 1 of the GO TO statement.	Execution sequence indicators are printed.
1N302	Tests format 2 of the GO TO statement.	Execution sequence indicators are printed.
1N303	Tests ALTER/GO TO combination.	Execution sequence indicators are printed.
1N304	Tests the DISPLAY statement.	No printed results except the output of the statement itself.
1N305	Tests the ACCEPT statement.	The data that was read is printed.
1N307	Tests the NOTE statement and NOTE paragraph.	Execution sequence indicators are printed.
1N310	Tests the use of switch-status-names in IF statements.	Execution sequence indicators are printed.
1N311	Tests non-floating insertion characters.	Receiving fields are printed.
1N312	Tests the use of floating insertion and replacement characters.	Receiving fields are printed.
1N313 A, B	Tests the use of level numbers 01-10 REDEFINES, SYNCHRONIZED, JUSTIFIED, BLANK, USAGE, and mixed PICTURE of A, X, 9.	Receiving fields are printed. Because the operation of the SYNC clause makes the size of the result indeterminate no expected result is printed.
1N314	Tests the CURRENCY SIGN clause.	Receiving fields are printed.
1N315	Tests the DECIMAL POINT IS COMMA clause.	Receiving fields are printed.
1N316 A, B	Tests the Data Division abbreviations: SYNC, PIC, COMP, JUST.	Receiving fields are printed.

Test ID	Summary of Test	Breakdown of Printed Results
1N317	Tests 30 character data-name and procedure-name.	Correctness indicator is printed.
1N318	Tests 120 character literal.	Correctness indicator is printed.

Test ID	Summary of Test.	Breakdown of Printed Results
2N001	ADD AZERO-DS-05V05 0.5 TO WRK-DS-01V00 WRK-DS-05V00 ROUNDED WRK-DS-06V06.	Arithmetic result is printed.
2N002	ADD A05ONES-DS-05V00 A99-DS-02V00 A1BONES-DS-09V09 GIVING WRK-DS-09V09 ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
2N003	ADD AZERO-DS-05V05 999 TO WRK-DS-03V10 WRK-DS-0201P ROUNDED WRK-DS-03V00 ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
2N004	ADD CORRESPONDING GRP-FOR-ADD-CORR-1 TO GRP-FOR-ADD-CORR-R.	Arithmetic result is printed.
2N005	ADD CORRESPONDING GRP-ADD-SUB-CORR TO GRP-FOR-ADD-CORR-R ROUNDED.	Arithmetic result is printed.
2N006	ADD CORRESPONDING GRP-SUBTRACT-CORR-3 TO GRP-FOR-ADD-CORR-R ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
2N007	SUBTRACT AZERO-DS-05 -99.9 FROM WRK-DS-02V00 WRK-DS-1BV00 ROUNDED WRK-DS-09V09.	Arithmetic result is printed.
2N008	SUBTRACT AZERO-DS-05V05 -99.9 FROM WRK-DS-02V00 WRK-DS-18V00 WRK-DS-09V09 ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
2N009	SUBTRACT AZERO-DS-05V05 -999 FROM WRK-DS-03V10 WRK-DS-0201P ROUNDED WRK-DS-03V00 ON SIZE ERROR MOVE"1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).

Test ID	Summary of Test	Breakdown of Printed Results
2N010	SUBTRACT CORRESPONDING GRP-FOR-ADD-CORR-1 FROM GRP-FOR-ADD-CORR-R.	Arithmetic result is printed.
2N011	SUBTRACT CORRESPONDING GRP-ADD-SUB-CORR FROM GRP-FOR-ADD-CORR-R ROUNDED.	Arithmetic result is printed.
2N013	MOVE CORRESPONDING GRP-MOVE-CORR-1 TO GRP-MOVE-CORR-R.	Arithmetic result is printed.
2N015	SUBTRACT CORRESPONDING GRP-SUBTRACT-CORR-3 FROM GRP-FOR-ADD-CORR-R ON SIZE ERROR MOVE "1" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).

Test ID	Summary of Test	Breakdown of Printed Results
2N016	MOVE CORRESPONDING GRP-TO-MOVE-CORR-1 TO GRP-TO-MOVE-CORR-R.	Receiving fields are printed.
2N017	PERFORM TEST-2NUC-017-A UNTIL TEST-2NUC-COND-99.	Execution sequence indicators are printed.
2N018	PERFORM TEST-2NUC-018-A VARYING WRK-DS-02V00 FROM 1 BY 1 UNTIL TEST-2NUC-COND-99.	Execution sequence indicators are printed.
2N019	PERFORM TEST2NUC-018-A VARYING WRK-DS-02V00 FROM A02TWOS-DS-02V00 BY A02TWOS-DS-02V00 UNTIL (WRK-DS-02V00 + 12) = 100.	Execution sequence indicators are printed.
2N020	Tests PERFORM ... VARYING to three levels.	Execution sequence indicators and final values of identifiers are printed.
2N022	Tests PERFORM ... VARYING to three levels; uses index-names.	Execution sequence indicators and final values of identifiers are printed.
2N023	Tests ALTER with the series option.	Execution sequence indicators are printed.

Test ID	Summary of Test	Breakdown of Printed Results
2N024	COMPUTE WRK-DS-02V00 = -9.	Arithmetic result is printed.
2N025	COMPUTE WRK-DS-02V00 = A99-DS-02V00.	Arithmetic result is printed.
2N026	COMPUTE WRK-DS-18V00 = A18ONES-DS-18V00 + A18ONES-DS-18V00.	Arithmetic result is printed.
2N027	COMPUTE WRK-DS-18V00 = A18TWOS-DS-18V00 - A18ONES-DS-18V00.	Arithmetic result is printed.
2N028	COMPUTE TALLY = 3 * A02TWOS-DU-02V00.	Arithmetic result is printed.
2N029	COMPUTE WRK-DS-05V00 = A02TWOS-DU-02V00 / A02TWOS-DS-03V02.	Arithmetic result is printed.
2N030	COMPUTE WRK-DS-05V00 = 3 ** ATWO-DS-01V00.	Arithmetic result is printed.
2N031	COMPUTE WRK-DS-02V00 ROUNDED = A99-DS-02V00 + AZERO-DS-05V05 - 2.5.	Arithmetic result is printed.
2N032	COMPUTE WRK-DS-02V00 = A99-DS-02V00 + AZERO-DS-05V05 ON SIZE ERROR MOVE "0" TO WRK-XN-00001.	Arithmetic result is printed; the last digit indicates whether SIZE ERROR worked correctly (1 = correct; 0 = incorrect).
2N033	Test REMAINDER option of DIVIDE.	Arithmetic result is printed.
2N034	COMPUTE WRK-DS-0201P ROUNDED = A05ONES DS-05V00 / 5 ON SIZE ERROR MOVE "1" TO WRK-XN-0001.	Arithmetic result is printed. The last digit indicates whether SIZE ERROR worked correctly (1=correct; 0=incorrect).

Test ID	Summary of Test	Breakdown of Printed Results
2N035 A, B	Using COMPUTE, tests a particular arithmetic expression both with and without parentheses.	Both results are printed.
2N036	Tests a particular arithmetic expression both with and without parentheses in IF statementa.	Results of comparison are printed.
2N037	Tests ADD with ON SIZE ERROR option in the true and false of an IF statement.	Resulte of arithmetic and comparisons are printed.
2N038	Tests IF statements nested on true and false paths of an IF statement.	Results of comparisons are printed.
2N039	Tests an IF statement containing a compound condition with one OR.	Results of the comparisons are printed.
2N040	Tests an IF statement containing a compound condition with one AND.	Resulta of the comparisons are printed.
2N041	Tests an IF etatement containing a compound condition with mixed ANDs and ORs.	Results of the comparisons are printed.
2N042	Tests an IF statement containing a compound condition with mixed ANDe, ORs, and NOTs.	Results of the comparisons are printed.
2N043	Tests IF statements uaing =, >, <, ae relational operators.	Reeults of the comparisone are printed.
2N045	Teste an IF statement ueing abbreviation 1 in the condition.	Results of the comparisons are printed.
2N046	Tests an IF atatement uaing abbreviation 2 in the condition.	Results of the comparisons are printed.

Test ID	Summary of Test	Breakdown of Printed Results
2N048	Tests unequal size operands fields in a conditional expression where equality should exist.	Results of the comparisons are printed.
2N049	Tests unequal size operand fields in a conditional expression where inequality should exist.	Results of the comparisons are printed.
2N050	Tests ACCEPT...FROM...	Data that was read is printed.
2N051	Tests DISPLAY literal UPON mnemonic-name.	No printed results except the actual output on the device specified by mnemonic-name.
2N052	Test DISPLAY mixed-literal-and-identifier-series UPON mnemonic-name.	No printed results except the actual output on the device specified by mnemonic-name.
2N053	Tests the operation of qualifications where it is required.	The fields to which reference is made are printed.
2N054	Tests the operation of qualification where it is not required.	The fields to which reference is made are printed.
2N056 A, B	Tests the operation of RENAMES without the THRU option.	The fields to which reference is made are printed.
2N057 A, B	Tests the operation of RENAMES with the THRU option.	The fields to which reference is made are printed.
2N058	Tests Data Division qualification and the plural form of the figurative constants.	The fields to which reference is made are printed.
2N059	Tests level 88 entries with the series option of the VALUE clause.	Results of comparisons are printed.
2N060	Tests the DATE-COMPILED paragraph.	No printed results at execution time.
2N061	Tests punctuation characters (; and ,) and all numeric procedure-names.	Correctness indicators are printed.

Test ID	Summary of Test	Breakdown of Printed Results
1TH01	Tests single level subscripting and the use of TALLY as a subscript.	Correctness indicators and table elements are printed.
1TH02	Tests single level indexing.	Correctness indicators and table elements are printed.
1TH03	Tests all combinations of operands in the SET...TO... statement.	Occurence numbers are printed.
1TH04	Tests the use of relation conditions containing indexes and index-data-items.	Results of the comparison are printed.

Test ID	Summary of Test	Breakdown of Printed Results
2TH01	Tests 2 and 3 level subscripting.	Correctness indicators and table elements are printed.
2TH02	Tests 2 and 3 level indexing.	Correctness indicators and table elements are printed.
2TH03	Tests the structural variations of format 1 of the SET statement that are unique to 2TBL.	Occurence numbers are printed.
2TH04	Tests the structural variations of format 2 of the SET statement.	Occurence numbers are printed.

Test ID	Summary of Test	Breakdown of Printed Results
3TH01	Tests the structural variations of format 1 of the SEARCH statement.	Correctness indicators and table elements found are printed.
3TH02	Tests the structural variations of format 2 of the SEARCH statement.	Correctness indicators and table elements found are printed.
3TH03	Same as 3TH01 but uses variable length tables.	Correctness indicators and table elements found are printed.
3TH04	Same as 3TH02 but uses variable length tables.	Correctness indicators and table elements found are printed.
3TH05	Creates a sequential file containing logical records whose description contains OCCURS with the DEPENDING option.	none
3TH06	Reads and verifies the file created in 3TH05.	Correctness indicators for every record and one for AT END are printed.

Test ID	Summary of Test	Breakdown of Printed Results
1SQ01	Writes an unblocked sequential tape file containing fixed length records.	no printed results.
1SQ02	Reads the file created in 1SQ01. Tests operation of the elements necessary for an input file; also validates 1SQ01.	One character for each record read: 1 = input record is equal to Working Storage item used to WRITE it. 0 = input record not equal. 9 = if EOF found prematurely, answer is filled with 9's at that point. One character for AT END: 1 = executed at correct time. 0 = not executed.
1SQ03	Writes a blocked sequential tape file containing fixed length records.	no printed results.
1SQ04	Reads the file created in 1SQ03. Tests operation of the elements necessary for an input file; also validates 1SQ03.	Same as 1SQ02.
1SQ05	Writes a blocked, multi-reel, sequential tape file containing fixed length records.	no printed results.
1SQ06	Reads the file created in 1SQ05. Tests the operation of these elements necessary for an input file; also validates 1SQ05.	Same as 1SQ02.
1SQ07	Writes an unblocked sequential tape file containing differing length records.	no printed results.
1SQ08	Reads file created in 1SQ07. Tests the operation of the elements necessary for an input file; also validates 1SQ07.	Same as 1SQ02.
1SQ09	Writes a blocked sequential tape file containing differing length records.	no printed results.
1SQ10	Reads the file created in 1SQ09. Tests the operation of the elements necessary for an input file; also validates 1SQ09.	Same as 1SQ02.
1SQ11	Writes a blocked, multi-reel, sequential tape file containing differing length records.	no printed results.
1SQ12	Reads the file created in 1SQ11. Tests the operation of the elements necessary for an input file; also validates 1SQ11.	Same as 1SQ02.

Test ID	Summary of Test	Breakdown of Printed Results
1SQ13	Writes an unblocked sequential mass storage file containing fixed length records.	no printed results.
1SQ14	Reads the file created in 1SQ13. Tests the operation of the elements necessary for an input file; also validates 1SQ13.	Same as 1SQ02.
1SQ15	Reads and updates the file created in 1SQ13. Tests the operation of the elements necessary for an input-output file.	no printed results.
1SQ16	Reads the file created in 1SQ13 and updated in 1SQ15. Validates 1SQ15.	Same as 1SQ02.
1SQ17	Writes blocked sequential mass storage file containing fixed length records.	no printed results.
1SQ18	Reads the file created in 1SQ17. Tests the operation of the elements necessary for an input file; also validates 1SQ17.	Same as 1SQ02.
1SQ19	Reads and updates the file created in 1SQ17. Tests the operation of the elements necessary for an input-output file.	no printed results.
1SQ20	Reads the file created in 1SQ17 and updated in 1SQ19. Validates 1SQ19.	Same as 1SQ02.
1SQ21	Writes a print file using BEFORE and AFTER ADVANCING.	Each line contains the statement used to write it.
1SQ23	Writes a blocked, multi-unit, sequential mass storage file containing fixed length records.	No printed results.
1SQ24	Reads the file created in 1SQ23. Tests the operation of the elements necessary for an input file; also validates 1SQ23.	Same as 1SQ02.
1SQ25	Reads and updates the file created 1SQ23. Tests the operation of the elements necessary for an input-output file in a multi-unit file environment.	No printed results.
1SQ26	Reads the file created in 1SQ23 and updated in 1SQ25. Validates 1SQ25.	Same as 1SQ02.
1SQ27-1SQ31	Tests the five options of RERUN in the order shown on page 82, items 44-48.	Execution sequence indicators are printed.

Test ID	Summary of Test	Breakdown of Printed Results
2SQ05	Tests the use of an OPTIONAL file.	One character for each record read: 1 = input record is equal to Working Storage item used to WRITE it. 0 = input record not equal. 9 = if EOF found prematurely, answer is filled with 9's at that point. One character for AT END; 1 = executed at correct time. 0 = not executed.
2SQ06	Creates a MULTIPLE FILE TAPE.	none
2SQ07	Reads and verifies the tape created in 2SQ06.	Same as 2SQ05.
2SQ08	Reads and verifies the last file on the tape created in 2SQ06.	Same as 2SQ05.
2SQ09	Creates a file for use in 2SQ10.	none
2SQ10	Tests the use of OPEN with the REVERSED option.	Same as 2SQ05.
2SQ11	Tests the ADVANCING option of WRITE using data-name and mnemonic name.	Each line contains the statement used to write it.
2SQ12	Tests the acceptability and operation of the USE statement.	Execution sequence indicators are printed.
2SQ13	Creates a file for use in 2SQ14 and tests "WRITE...FROM...".	None
2SQ14	Read and verifies the file created in 2SQ13; and tests "READ...INTO...".	Same as 2SQ05.
2SQ15	Tests LABEL RECORDS data-name together with LABEL PROCEDURE declaratives.	One character for each condition as follows: (1) 1 indicates BEGINNING LABEL on OUTPUT written. (2) 1 indicates ENDING LABEL on OUTPUT written. (3) 1 indicates BEGINNING LABEL on INPUT verified. (4) 1 indicates ENDING LABEL on OUTPUT verified.

Test ID	Summary of Test	Breakdown of Printed Results
1RA01	Writes an unblocked mass storage file in random access mode.	no printed results
1RA02	Reads and verifies the file created in 1RA01.	One character per record read: 1 = input record is equal to Working Storage item used to WRITE it. 0 = input record not equal. 9 = INVALID KEY exit taken on record.
1RA03	Reads and updates inpplace the file created in 1RA01.	no printed results.
1RA04	Reads and verifies the file created in 1RA01 and updated in 1RA03.	Same as 1RA02.
1RA05	Logically equivalent to 1RA01 but for blocked records.	no printed results.
1RA06	Logically equivalent to 1RA01 but for blocked records.	Same as 1RA02.
1RA07	Logically equivalent to 1RA03 but for blocked records.	no printed results.
1RA08	Logically equivalent to 1RA04 but for blocked records.	Same as 1RA02.

Test ID	Summary of Test	Breakdown of Printed Results
2RA01	Creates a random access file whose records are unblocked and of differing lengths. Uses data-name option of FILE-LIMITS clause, and "TO integer" option of BLOCK CONTAINS.	No printed results.
2RA02	Reads and verifies the file created in 2RA01.	One character per record read: 1 = input record is equal to Working Storage item used to WRITE it. 0 = input record not equal. 9 = INVALID KEY exit taken on record.
2RA03	Reads and updates in place the file created in 2RA01.	No printed results.
2RA04	Reads and verifies the file created in 2RA01 and updated in 2RA03.	Same as 2RA02.
2RA05	Logically equivalent to 2RA02 but for blocked records.	No printed results.
2RA06	Logically equivalent to 2RA02 but for blocked records.	Same as 2RA02.
2RA07	Logically equivalent to 2RA03 but for blocked records.	No printed results.
2RA08	Logically equivalent to 2RA04 but for blocked records.	Same as 2RA02.
2RA09 thru 2RA12	Logically equivalent to 2RA01 thru 2RA04 except that WRITE... FROM, READ... INTO, literal series option of FILE-LIMITS clause, STANDARD option of LABEL RECORDS clause and literal-series option of VALUE OF clauses are used.	Same as 2RA01 thru 2RA04.

Test ID	Summary of Test	Breakdown of Printed Results
2RA13 thru 2RA16	Logically equivalent to 2RA05 thru 2RA08 except that WRITE... FROM, READ...INTO, data-name series option of FILE-LIMITS clause, data-name series option of LABEL RECORDS clause, data-name series of VALUE OF clause, and in 2RA16 CLOSE...WITH LOCK, are used.	Same as 2RA05 thru 2RA08.
2RA17	Tests the acceptability and operation of the USE statement.	Execution sequence indicators are printed.
2RA18	Tests LABEL RECORDS data-name in conjunction with USE...LABEL PROCEDURE ON OUTPUT, INPUT, I-O.	Results indicate a '1' for 1. OUTPUT HEADER written 2. OUTPUT TRAILER written 3. INPUT HEADER verified 4. INPUT TRAILER verified 5. I-O HEADER verified 6. I-O TRAILER verified.

Test ID	Summary of Test	Breakdown of Printed Results
1ST01	Sorts 6 records created in and checks the sequence of the returned records in the OUTPUT PROCEDURE.	aaabbbcccdddeeefffg* Each returned record, represented by a through f, has three tests made upon it: for content of first non-key field; the sort key; for the content of the second non-key field. The final g is 0 if an incorrect number of records are returned.
1ST02	Services 1ST03 by creating an input for it. No sort.	No printed results.
1ST03	The file created by 1ST02 is sorted descending, with the OUTPUT PROCEDURE checking sequence on keye and contents of non-key areas. Sorted records are written onto a FILE-NAME-USING-4 for paseing to 1ST04.	aabbcc---xxyyz* Each of the records, represented by a through y, has two tests made upon it: for the non-key area; for the key area. z represents 1 for good count, 0 for bad count.
1ST04	The USING file is the descendingly sorted file of 1ST03 which ie sorted into ascending order and passed to 1ST05.	No printed resulte.
1ST05	Checks sequence of file paeeed from 1ST04.	Same as 1ST03.
1ST06	Sorts with THRU option of INPUT PROCEDURE and with the OR option of the ASSIGN clause. Depends upon blank and zero relative collating eequences for SORT and IF being the same. GIVING used.	No printed results.
1ST07	Checke eequence of file paeeed from 1ST06.	aaabbbccc...xxxyyyz* Each of the keys is separately checked in each of the records. Correct count is reflected in z.
1ST08	Sort with multiple RELEASE statements in the INPUT PROCEDURE and multiple descending keys. OUTPUT PROCEDURE used.	aaabbbccc...xxxyyyz* Each of the keys is separately checked in each of the records. Correct amount is reflected in z.
1ST09	Sorte with multiple RETURN statements in the OUTPUT PROCEDURE, on eight types of aecending Keys. Uses INPUT PROCEDURE.	abc...xyz* Records are checked only for the most recently changed Key. The record count is reflected in z.

Test ID	Summary of Test	Breakdown of Printed Results
1ST10	Creates a file of differing length records to be passed to 1ST11.	No printed results.
1ST11	Sorts file from 1ST10 records, descending, on two series Keys. Uses USING, GIVING and the TO option of RECORD CONTAINS.	No printed results.
1ST12	Checks sequence of file passed from 1ST11.	aaabbbccc...xxxyyyz* Each record is checked on three fields: a body field unique to each record length; Key-1; and Key-2. The record count is reflected in z.
1ST13	Prepares a 3 reel tape file (using CLOSE REEL) each containing 26 records of 33 characters each to be passed to 1ST14.	No printed results.
1ST14	Sorts the multi-reel file created in 1ST13 into descending sequence. Uses USING, GIVING. The sort Key is an 01 elementary item and is also named in the DATA RECORDS clauses of the SD.	No printed results.
1ST15	Checks sequence of the file passed from 1ST14.	abc...xyz* The 78 records are each checked for total contents once. The record count is reflected in z.

Test ID	Summary of Test	Breakdown of Printed Results
2ST01	A double sort in which one sort (I/P, GIVING) is by another (USING O/P) in which the USING file is the same FD-name as is the previous GIVING file.	aabbcc...xxyyz* Each of the records returned is checked first on an indicative non-key field and then upon the Key field. The total record count is reflected in z.
2ST02	SAME RECORD clause for both an SD and two FD files, all defined within this test.	aaabbbccc...xxxyyyz* The records (here represented by a,b,c,...x,y) are checked: upon a set non-key field; on another non-key field; and on the sort Key. The record count is reflected in z.
2ST03	The USE of SAME SORT and SAME RECORD clauses and of multiple SAME clauses. Method is generally the same as for the 2ST02 except for the value of Keys generated and expected after sorting.	aaa bbb ccc ... xxx yyy z* The records are checked: on an alphabetic justified non-key field; on another, similar, non-key field; and on the unsigned numeric display Key field. The record count is reflected in z.
2ST04	The FROM option of the RELEASE statement and the RESERVE clause of the SELECT statements. Uses INPUT PROCEDURE and GIVING. After the sort, it reads back and checks the sorted records.	aaa bbb ... xxx yyy z* The records are checked: on the sort-key; on a fixed numeric field; and on a fixed alphanumeric field. The record count is reflected in z.
2ST05	Tests the INTO option of RETURN along with dual reference capability of the returned item. Uses both INPUT and OUTPUT PROCEDURE's.	aaabbbccc...xxxyyyz* The records are checked: to verify that the W-S item referenced by the INTO clause is identical to the sort record returned; and the key is checked for expected contents. The record count if reflected in z.
2ST06	Checks conformity of the RELEASE and RETURN statements to the rules of group MOVES.	aaaabbbbcccc...xxxxyyyyz* The records returned (here represented as a,b,c,..x,y, where a represents two consecutive records, b two consecutive records, etc.) are compared thusly: the Sort-record returned versus the W-S item named in the INTO clause; the next Sort-record to be returned versus a separate W-S item into which the first Sort-record has been MOVED. a sequence check on the record in the W-S item named in the INTO clauses; and a sequence checks on the Sort-record itself.

Test ID	Summary of Teet	Breakdown of Printed Results
1RW01	Teets a minimal Report Writer environment.	The results are reports. These reports are validated by comparing them to the "standard" eet of reports which are printed conventionally from the tests.
1RW02	Teste the following: 1) HEADING, FOOTING clausee, with minimal environment, 2) Page Counter with size determined by the PICTURE of SOURCE item, 3) How Line Counter effecte PAGE-LIMIT and NEXT GROUP, 4) Report Group, LINE NUMBER with PLUS option, NEXT GROUP clause, PLUS option, and 5) Report Elementary, JUSTIFIED RIGHT: qualification by RD, SOURCE IS Working-Storage-item.	The resulte are reports. These reports are validated by comparing them to the "etandard" set of reports which are printed conventionally from the tests.
1RW03	Teets the following: 1) Report Elementary; LINE with absolute values, PLUS, and NEXT PAGE, 2) NEXT GROUP with absolute values and NEXT PAGE, 3) LINE, 4) TYPE, 5) NEXT GROUP with epecial casee in other than groups, 6) ability to have non-contradicting LINE clauees at different levele, 7) abbreviations RH and RP and 8) JUSTIFIED RIGHT clause.	The results are reports. These reports are validated by comparing them to the "standard" set of reports which are printed conventionally from the tests.

Test ID	Summary of Test	Breakdown of printed results
2RW01	Tests Controls would have a minimal environment consisting of the clauses; 1) RD; CONTROL IS identifier, PAGE LIMITs Clauses, FOOTING option of PAGE LIMIT, 2) Report Group clauses; CF option of TYPE, 3) Report Elementary clauses; GROUP INDICATE AND SUM. GROUP INDICATE is tested for page break without a control break.	The results are reports. These reports are validated by comparing them to the "standard" set of reports which are printed conventionally from the tests.
2RW02	Tests the following: 1) suppression of items by omitting their COLUMN clause, 2) Multiple DETAIL lines, 3) SUM data-name series, 4) SUM using sum-counter-name, 5) SUM UPON 6) SUM RESET 7) SUM UPON RESET 8) TERMINATE followed by INITIATE, 9) summary reporting.	The results are reports. These reports are validated by comparing them to the "standard" set of reports which are printed conventionally from the tests.
2RW03	Writes multiple reports on one file (series option of REPORTS ARE clause), WITH CODE clause, series option of TERMINATE, single and multiple character CODE, referencing SUM counter from Procedure Division, multiple GENERATEs for a given elementary item. A high level SUM with no low level SUM is included in this test.	The results are reports. These reports are validated by comparing them to the "standard" set of reports which are printed conventionally from the tests.
2RW04	Tests the USE BEFORE REPORTING declarative, to see that a change just before GENERATE is reflected by the USE declarative.	The results are reports. These reports are validated by comparing them to the "standard" set of reports which are printed conventionally from the tests.
2RW05	Teste the referencing of an out-of-line USE BEFORE REPORTING declarative by an in-line PERFORM. All line references are absolute.	The results are reports. These reports are validated by comparing them to the "standard" set of reports which are printed conventionally from the tests.

Test ID	Summary of Test	Breakdown of Printed Results
1SG01	ALTER in 11 section GO TO in 00 section PERFORMing 00 from 11	Execution sequence indicators are printed.
1SG02	ALTER in 00 section GO TO in 00 section PERFORMing 00 from 00	Execution sequence indicators are printed.
1SG03	ALTER in 50 section GO TO in 50 section PERFORMing 50 from 49	Execution sequence indicators are printed.
1SG04	ALTER in 50 GO TO in 50 PERFORMing 50 from 50	Execution sequence indicators are printed.
1SG05	ALTER in 66 GO TO in 33 PERFORMing 33 from 66	Execution sequence indicators are printed.

Test ID	Summary of Test	Breakdown of Printed Results
1SG11	ALTERs in 12 GO TOs in 01, 02 PERFORMing 01 thru 02 from 12	Execution sequence indicators are printed.
1SG12	ALTERs in 55 GO TOs in 55, 55 PERFORMing 55 thru 55 from 14	Execution sequence indicators are printed.
1SG13	ALTERs in 70 GO TOs in 07, 08 PERFORMing 07 thru 08 from 70	Execution sequence indicators are printed.
1SG14	ALTERs in 60 GO TOs in 60 PERFORMing 60 thru 60 from 60	Execution sequence indicators are printed.

Test ID	Summary of Test	Breakdown of Printed Results
2SG01 A B C	Uses tests 1SG02,1SG03, and 1SG05 with SEGMENT LIMIT 30.	Printout of results from 1SEG.
2SG02 A B C	Uses tests 1SG11,1SG12,1SG13 with SEGMENT LIMIT 9.	Printout of results from 1SEG.

Test ID	Summary of Test	Breakdown of Printed Results
1LB01	Compares copied versus a non-copied data item.	One character for each comparison. 1 if found to be as expected, 0 if not found to be as expected.
1LB02	Copied paragraphs move data items and check for expected contents.	One character for each comparison, 1 if found to be as expected, 0 if not found to be as expected.
1LB03	FILE-CONTROL section is copied.	One character for each comparison, 1 if found to be as expected, 0 if not found to be as expected.
1LB04	A minimal report writer environment contains copied RD entry.	The results are reports. These reports are validated by comparing them to the "standard" set of reports which are printed conventionally from the tests.
1LB05	A sequential file is written, read, and compared, using copied FD and 01 entries.	One character for each comparison. 1 if found to be as expected, 0 if not found to be as expected.
1LB06	A minimal sort with INPUT and OUTPUT procedures is executed using a copied SD entry.	aabbcc...xxyyz* Each of the records returned is checked first on an indicative non-key field and then upon the Key field. The total record count is reflected in z.
1LB08	SOURCE COMPUTER and OBJECT COMPUTER are copied.	The test program listing must be checked to validate results.
1LB09	Switch status's in SPECIAL-NAMEs are copied.	One character for each comparison. 1 if found to be as expected, 0 if not found to be as expected.
1LB10	Copies I-O-CONTROL SAME AREA clause. Writes and reads back two sequential files.	One character for each comparison. 1 if found to be as expected, o if not found to be as expected.

Test ID	Summary of Test	Breakdown of Printed Results
2LB01	Compares copied vereus a non-copied data item with REPLACING option used.	One character for each comparison. 1 if found to be as expected, 0 if not found to be as expected.
2LB02	Copied paragraphs move replaced data items and check for expected contents.	One character for each comparison, 1 if found to be as expected, 0 if not found to be as expected.
2LB03	FILE-CONTROL section is copied with the file name replaced.	One character for each comparison, 1 if found to be as expected, 0 if not found to be as expected.
2LB05	A sequential file is written, read, and compared, ueing FD and 01 entries copied with the REPLACING option.	One character for each comparison. 1 if found to be as expected, 0 if not found to be as expected.
2LB08	SOURCE-COMPUTER and OBJECT-COMPUTER names are copied with the REPLACING option.	The test program listing must be checked to validate results.
2LB09	SPECIAL-NAMES is copied with the switch status's replaced.	One character for each comparison. 1 if found to be as expected, 0 if not found to be as expected.
2LB10	Copiee I-O CONTROL SAME AREA clause REPLACING the file names. Writes and reads back two sequential files.	One character for each comparison. 1 if found to be as expected, 0 if not found to be as expected.

APPENDIX V

PFM DIAGNOSTIC MESSAGES.

The following is a list of diagnostic messages with their respective interpretation:

Diagnostic Message	Explanation
CARD-RDR MANDATORY CARD 24 INFO MANDATORY	Environment table has missing specification for CARD-RDR.
COMPUTER-NAME MANDATORY CARD 1 INFO MANDATORY CARD 2 INFO MANDATORY	Environment table has missing COMPUTER-NAME. Card numbers refer to card images generated for the Population File.
ERROR IN FIRST 2 COLUMNS	A control card was expected. Chances are that this is a data card for a previous aborted action.
FIELD 28 BLANK, ASSUMED D	Environment Table input card number 9 has a blank in field 28. D is assumed.
FIELD 36 BLANK, ASSUMED D	Environment Table input card number 10 has a blank in field 36. D is assumed.
INVALID ACTION	Header card encountered with an invalid action code. The valid action codes are A, D, C & P.
MATCH-OLD ENTRIES DELETED	On an add action if a header match occurs the old entries will be deleted and the new entries will replace them.
NO MATCH	Means old Population File Master is at EOF and no matching header was found. In the case of an add this may not be an error.

Diagnostic Message (cont'd.)	Explanation (cont'd.)
NO MATCH--SO WILL IGNORE	On a change action if no matching header is found the action and subsequent change cards will be ignored.
PRINTER-NAME MANDATORY CARD 25 INFO MANDATORY	Environment table has missing specification for PRINTER-NAME.
SEQ ERROR IN ENV ON POP FILE ADVISE DELETING, THEN ADDING NEW	Found sequence error on Population File Master within an Environment Table. This should never occur but may occur as the result of a tape read error.
TAPE-UNIT-1 MANDATORY CARD 19 INFO MANDATORY	Environment table has missing specification for TAPE-UNIT-1.
TAPE-UNIT-2 MANDATORY CARD 20 INFO MANDATORY	Environment table has missing specification for TAPE-UNIT-2.
TAPE-UNIT-3 MANDATORY CARD 21 INFO MANDATORY	Environment table has missing specification for TAPE-UNIT-3.
USABLE-SIZE MANDATORY CARD 4 INFO MANDATORY	Environment table has missing USABLE-SIZE.

APPENDIX VI

CONSIDERATIONS IN CREATING TESTS

SEQUENCE NUMBERING

Sequence numbers (column 1-6) must be assigned to statements within each test in accordance with the algorithm shown in Figure 6.

Three points must be noted:

1. Phase 3 of the Selector inserts Division Headers and certain section and paragraph-names on the basis of these sequence numbers, and therefore, misnumbering will cause disorganization in the Selector output.

2. Phase 3 of the Selector program checks for matching sequence numbers in adjacent cards. When this condition is detected, one of the matching pair will be deleted. A diagnostic will be issued if the deleted card did not match the other in columns 7-72. Therefore, care must be taken in sequence numbering tests that appear in the same program (see the assignments in 3.1.1.2.1). Duplicate numbers should be assigned to only those items that are, in fact, duplicates.

3. Most of the available serial numbers have been exhausted by existing tests. The numbers remaining in each series can be ascertained by checking the listing of the highest numbered test in each module.

DIVISION	SECTION	CARD COLUMN 1	2	3	4	5	6
IDENT.	PROG-ID	0	0	0	1	00-99 SEQUENTIAL NO.	
	AUTHOR	0	0	0	2		
	INSTAL	0	0	0	3		
	DATE-WR	0	0	0	4		
	DATE-COMP	0	0	0	5		
	SECUR	0	0	0	6		
	REMARKS	0	0	0	7		
ENVIR.	CONFIG	0	0	0	8		
	FILE-CONTROL	0	FUNCTION	0000-9999 SEQUENTIAL NUMBER			
	I-O-CONTROL	1	0 NUC 1 TBL	0 RERUN 1 SAME 2 MULTIPLE	000-999 SEQUENTIAL NUMBER		
DATA	FILE SECTION	2,3	2 SEQ 3 RAC	0000-9999 SEQUENTIAL NUMBER			
	WORKING-STORAGE (77)	4	5 SRT 6 RPW				
	(OTHERS)	5	7 SEG				
	REPORT SECTION	6	8 LIB				
PROC.	DECLARATIVES	7	9 SUPPORT				
	NON-DECLAR.	8,9					

NOTE: Sequential number field may restart at zero whenever a digit to its left changes

Figure 6. Sequence Numbering for Test Cards

TEST REFERENCE TO ENVIRONMENTAL DATA

Test routine cards that are to be replaced by environmental data entries have a special format. Column 7 contains an 'E'. Columns 8 - 25 may contain the drop range indicator:

FROMmmmmmmTOnnnnnn

This indicates that cards from this test with sequence numbers mmmmmm, nnnnnn and all intervening cards will be eliminated by the Selector if the environmental entry whose sequence number appears in columns 75 - 80 of this card has an N in its column 7 (indicating that the information required is not available for this configuration). Column 74 contains a D if the entire test is to be eliminated by the Selector if the environmental data is absent. (This indicator overrides columns 8 - 25). In the absence of either of these indicators, only the requesting card itself will be dropped if the environmental data is missing. Column 75 - 80 contains the sequence number of the entry in the environmental data set that is to replace col 7 - 80 of this entry.

For example, consider the following cards from test 1N022, each of which requests replacement by an environmental entry:

```
1      7                                          74     80
T∆1N022...
    :
400124E                                            ET0040
400136EFROM400134TO400145                          ET0300
400181E                                           DET0050
```

Card 400124 requests that the environment data card with sequence number ET0040 replace it from col 7 through 80. If card ET0040 contains an N in its column 7, card 400124 will be omitted from the test by the Selector program. Card 400136 requests replacement by environment entry ET0300. If ET0300 has an N in its column 7, the Selector will omit cards 400134-400145 from test 1N022. Card 400181 requests replacement by card ET0050. If ET0050 has an N in its column 7, test 1N022

will be omitted in its entirety by the Selector program. In any case, when the requested environmental data card has an N in its column 7, a diagnostic message is issued.

DATA NAMING CONVENTION

Data items that are used as source and receiving items for test statements are assigned names under the following naming convention:

```
{ {WRK}{-{CU}}      {-{Pnnnn}}  [-Ønnn {D}] [-S] [-{L}] [-nnnn] }
  {value}{ {CS}     {-{nnnnP}}          {F}        {R}
           {DU}     {-{nnVnn}}
           {DS}      {nnnnn}
           {IX}
           {XJ}
           {XN}
           {AJ}
           {AN}
 {AE}
 {NE}  -nnnn
 GRP-name
```

The uppermost format is used for all elementary items except those with editing symbols within their PICTURE. Fields subject to change begin with the designation 'WRK' while constant source items begin with a mnemonic or literal indication of their VALUE. Because 1NUC requires that data-names begin with an alphabetical character, the names assigned to source items containing numeric literals as value will begin with the character 'A'.

The next letters indicate:

CU	unsigned COMPUTATIONAL
CS	signed COMPUTATIONAL
DU	unsigned DISPLAY
DS	signed DISPLAY
IX	index items
XJ	alphanumeric JUSTIFIED
XN	alphanumeric unjustified
AJ	alphabetic JUSTIFIED
AN	alphabetic unjustified

The next five characters indicate the size and point location. P indicates the scaling P, either left or right, and when it appears, the two digits nearest it indicate the number of P's while the other two digits indicate the number of 9's in the item's PICTURE. V indicates an implied decimal point. The digits indicated by n give the number of X's, 9's, or A's in the PICTURE. The optional Ønnn indicates an OCCURS clause with nnn occurrences. The trailing D indicates a DEPENDING clause while F indicates the lack of same. S indicates an item containing a REDEFINES clause, L or R indicate SYNCHRONIZED LEFT and RIGHT respectively and the final nnnn provides for a unique serial number.

The lower two formats, one for edited items (both alphanumeric and numeric) and one for group items are self explanatory. This convention serves two purposes. First, it yields a self-documenting data-name. Second, and more important, it enables the Selector Program to identify elementary non-report level 77 items with identical data descriptions through a simple comparison of data-names. When the result of this comparison is equal, the Selector eliminates from each test program all but one of each set of identically defined level 77 data items. Elimination may be suppressed by adding to the data-name of a given item a unique serial number, shown as the final nnnn in the preceding format.

The Selector program also eliminates all but one from each set of FD's and 01's with identical names and descriptions. When an FD is eliminated, the FD clause itself, its defining clauses (RECORDING MODE, BLOCK etc.), and the definitions of all its records are dropped. When a level 01 item is eliminated all items subordinate to it are also eliminated. The names of items with level numbers 02-49 are not examined on an individual basis, and these items are only eliminated when the level 01 to which they are subordinate is eliminated.

Because each Test includes all the file and data definitions it requires, this process of elimination is essential in generating a test program of reasonable size.

USE OF THE SUPPORT ROUTINE

The Support Routine compares the result achieved by each test with the expected result. If the character by character comparison indicates a difference, the results are printed with an underline. If 'ALL' is stated on the HDWR card (see 3.2.1) all results are printed.

In order to utilize the routine, the following procedure steps are required:

1. MOVE results TO SUP-WK-A.

   SUP-WK-A is a 120 character area used to hold results. If more than one result is to be moved, each can be moved character by character to SUP-WK-B under control of a subscript.

2. MOVE 'expected-result*' TO SUP-WK-C.

   Expected results can be most easily stated in a literal. Note that the result is terminated by the additional character '*'. This character limits the comparison between SUP-WK-A and SUP-WK-C.

3. MOVE 'test-name' TO SUP-ID-WK-A.

   Test-names are five characters in length and are discussed in 3.1.1.2.1

4. PERFORM SUPPORT-RTN THRU SUP-RTN-C.

   Return is made to the next statement.

Variations of this standard linkage may be required by results that exceed 120 characters, or by other unusual circumstances. Perusal of the test listings will indicate various alterations that can be used.

APPENDIX VII

SYSTEM GENERATION SPECIFICATIONS

Required Configuration

Any IBM System 360 with the following minimum configuration may be used:

1. a console typewriter with unit address 01F
2. a card punch with unit address 00D
3. two 9-track 800 bpi tape units with unit addresses 182 and 183.

Procedure

Mount input tape on tape unit 182. Ready a scratch tape (for the population file) on tape unit 183. Ready the card punch (00D). Set the machine load address to unit 182 and press LOAD. The program will punch an object deck of the CCVS Character Code Conversion program and source decks for the three phases of the CCVS Selector Program. It will then copy the Population File onto the tape readied on unit 183. The user may now separate the punched output into four decks:

1. the character code conversion program loadable deck
2. the Selector phase 1 source deck
3. the Selector phase 2 source deck
4. the Selector phase 3 source deck.

Each of these decks is preceded by a special header card to make it easy to find. Each header contains an identification in columns 1-12 for the deck following it (TRANSLATOR, SEL PHASE 1, SEL PHASE 2, SEL PHASE 3). Columns 13-80 contain 12-9 punches making the headers easy to locate in the uninterpreted deck.

The Character Code Conversion Program may now be used to translate the new population file to the desired character set. The Selector programs may be compiled and used to select the other CCVS utility programs and tests.

NORMAL End of Job Message

END OF JOB - Normal end of job has been reached.

ABNORMAL End of Job Messages

1. I/O ERROR TAPE1 - An error occurred reading the input tape on 182.
2. I/O ERROR TAPE2 - An error occurred writing the output tape on 183.
3. I/O ERROR PUNCH - An error occurred punching the output.

In all cases, rerun job.

APPENDIX VIII

SAMPLE CONTROL CARDS FOR CHARACTER CONVERSION PROGRAM

The following unusual characters appear in the control cards:

character	card code
¦	12-7-8
¬	11-7-8
¢	12-2-8
↑	12-0
_	0-5-8
:	2-8

For illustrative purposes, all of the cards shown below contain the complete range of characters that differ among the four implementations. In actual use, a pair of SOURCE/OBJECT cards need only contain those characters that differ between the two implementations in question. For example, the cards for the IBM/360 to B-3500 conversion need only contain the quote (' to ").

The sample control cards are as follows:

IBM/360

```
S Ø U R C E 1 8 1 C 8 ( ) ; + = > < ' &
```

CDC-6400

```
Ø B J E C T 1 8 0 A 8 % ¤ ¦ & # ¬ ¢ @ &
```

GE-635

```
Ø B J E C T 1 8 0 A 8 ( ) ; ↑ _ = + > &
```

B-3500

```
Ø B J E C T 1 8 2 C 8 ( ) ; + = > < " &
```

U-1108

```
Ø B J E C T 1 8 0 A 8 % ¤ ; & # = + @ :
```

Security Classification

DOCUMENT CONTROL DATA - R & D

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

1. ORIGINATING ACTIVITY *(Corporate author)* Directorate of Systems Design & Development Hq Electronic Systems Division L. G. Hanscom Field, Bedford, Mass. 01730	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP N/A

3. REPORT TITLE

USER'S MANUAL COBOL COMPILER VALIDATION SYSTEM

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

None

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

None

6. REPORT DATE July 1970	7a. TOTAL NO. OF PAGES 144	7b. NO. OF REFS
8a. CONTRACT OR GRANT NO. IN-HOUSE b. PROJECT NO. 6917 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-70-274 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	

10. DISTRIBUTION STATEMENT

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

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Hq Electronic Systems Division (AFSC) L. G. Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

This technical report consists of detailed specifications for the use of the COBOL Compiler Validation System (CCVS). The system is designed to measure the compliance of a specific COBOL compiler against the American National Standards Institute standard COBOL (ANSI X3.23-1968). This report describes the card input formats, deck structures, tape requirements, test modules, and operator procedures required to use the system.

DD FORM 1 NOV 65 1473

Security Classification

Security Classification

14. KEY WORDS	LINK A ROLE	LINK A WT	LINK B ROLE	LINK B WT	LINK C ROLE	LINK C WT
COBOL compiler validation						

Security Classification