A Practical Application to Data Consistency Validation Using Finite State Automata

Michael Lee Richardson
Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses

Part of the Computer Sciences Commons

Recommended Citation
http://scholarworks.wmich.edu/masters_theses/1454

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
A PRACTICAL APPLICATION TO DATA CONSISTENCY VALIDATION USING FINITE STATE AUTOMATA

by

Michael Lee Richardson

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
August 1985

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
A PRACTICAL APPLICATION TO DATA CONSISTENCY VALIDATION USING FINITE STATE AUTOMATA

Michael Lee Richardson, M.S.
Western Michigan University, 1985

The data which are input to a complex information system must often be verified for both syntactic and semantic consistency. The syntactic consistency can be enforced in a straightforward manner during the data entry process. The semantic inconsistencies can be more difficult to discover. A general approach, using finite state automata, is demonstrated here as applied to enforce consistency on client questionnaire responses in a large information system. This approach is developed using those software engineering standards deemed appropriate for this information system. The standards are outlined and justified in this research. A discussion of semantic inconsistencies which require more sophisticated techniques is included.
ACKNOWLEDGMENTS

I would like to express my appreciation to Dr. Dionysius Kountanis for his support and encouragement in the completion of this thesis. Dr. Kountanis has been of great help and assistance to me in attaining this degree. I would also like to thank Mr. Robert Trenary for his review of this research and the valuable insight he provided me with throughout the development of this paper. Completion of this thesis would not have been possible without the loving support of my wife, Kara, and our children, Laura and Katie.

Michael Lee Richardson
INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
Richardson, Michael Lee

A PRACTICAL APPLICATION TO DATA CONSISTENCY VALIDATION USING
FINITE STATE AUTOMATA

Western Michigan University

University Microfilms International 300 N. Zeeb Road, Ann Arbor, MI 48106

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs or pages ☑
2. Colored illustrations, paper or print ☐
3. Photographs with dark background ☑
4. Illustrations are poor copy ☐
5. Pages with black marks, not original copy ☑
6. Print shows through as there is text on both sides of page ☑
7. Indistinct, broken or small print on several pages ✓
8. Print exceeds margin requirements ☐
9. Tightly bound copy with print lost in spine ☐
10. Computer printout pages with indistinct print ☐
11. Page(s) _________ lacking when material received, and not available from school or author.
12. Page(s) _________ seem to be missing in numbering only as text follows.
13. Two pages numbered _______. Text follows.
14. Curling and wrinkled pages ☐
15. Dissertation contains pages with print at a slant, filmed as received _______
16. Other ________________________________

____________________________________
____________________________________

University Microfilms International

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>I. INFORMATION SYSTEMS</td>
<td>1</td>
</tr>
<tr>
<td>Uses of Information</td>
<td>3</td>
</tr>
<tr>
<td>System Verification</td>
<td>4</td>
</tr>
<tr>
<td>Summary</td>
<td>8</td>
</tr>
<tr>
<td>II. RELATED RESEARCH</td>
<td>10</td>
</tr>
<tr>
<td>Review of Literature</td>
<td>11</td>
</tr>
<tr>
<td>III. FINITE STATE AUTOMATON</td>
<td>17</td>
</tr>
<tr>
<td>FSA Representation</td>
<td>18</td>
</tr>
<tr>
<td>FSA Implementation</td>
<td>19</td>
</tr>
<tr>
<td>The Moore Machine</td>
<td>20</td>
</tr>
<tr>
<td>Balanced FSAs</td>
<td>21</td>
</tr>
<tr>
<td>IV. APPLYING FSA TECHNIQUES TO ADES</td>
<td>28</td>
</tr>
<tr>
<td>Working Environment</td>
<td>29</td>
</tr>
<tr>
<td>The Problem</td>
<td>31</td>
</tr>
<tr>
<td>Software Engineering</td>
<td>33</td>
</tr>
<tr>
<td>Documentation Standards</td>
<td>34</td>
</tr>
<tr>
<td>System Design</td>
<td>36</td>
</tr>
<tr>
<td>The Solution</td>
<td>38</td>
</tr>
<tr>
<td>Implementation</td>
<td>46</td>
</tr>
<tr>
<td>Implementation Results</td>
<td>50</td>
</tr>
</tbody>
</table>
Table of Contents—Continued

CHAPTER

V. CONCLUSIONS AND RECOMMENDATIONS 53
   Recommendations for Further Research 54

APPENDICES

A. ADES Admission Forms 58
B. ADES Discharge Forms 61
C. System Requirements Study 64
D. Inconsistency Check List 75
E. Programming Standards 78
F. Modular Definitions 101
G. Hierarchy Plus Input-Process Output (HIPO) 108
H. State Table 135
I. INCON Software 139

BIBLIOGRAPHY 182
**LIST OF FIGURES**

1. Development Process ........................................ 6
2. Testing Format ............................................... 23
3. Sample balanced FSA ......................................... 25
4. Reduction of Balanced FSA .................................... 25
5. Minimization Steps ........................................... 26
6. Minimized FSA ................................................ 26
7. Transition Diagram for Form 551 ............................. 42
8. Transition Diagram for Forms 551/552 ........................ 43
9. Transition Diagram for Form 552 ............................. 43
10. Transition Diagram for Form 651 ............................. 44
11. Transition Diagram for Forms 651/652 ........................ 44
12. Transition Diagram for Form 652 ............................. 44
13. Transition Diagram for Forms 552-652 ........................ 45

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER I

INFORMATION SYSTEMS

Information is the essential ingredient in any decision-making process. The need for improved information systems in recent years has been made critical by the steady growth in size and complexity of organizations and data.

Freeman (1975) defined information to be an organized collection of facts derived from a variety of sources. This definition of information led Freeman to conclude that an information system is a data structure with access. This access has two primary functions: storage and maintenance of representations of structured information, and the presentation of structured output from the stored representations.

This definition of information systems excludes any complex processing of information, indicating these systems to be identical to a database system. While complex operations may be supported by the information system, the functions as presented by Freeman (1975) involve rather simple operations.

Horton (1974) defined an information system as the fundamental tool with which one is ultimately concerned. The core of such a system is composed of steps taken to collect, process, and display information needed by the user or decision maker. Collectively, Horton considered these steps as the components of the information system.
Burch and Stratert (1973) pointed out that information systems may be formal or informal in nature. They characterized informal information systems as somewhat spasmodic and based upon general readings and occasional studies. Realizing that most large systems must communicate to their users in a formal manner, Burch and Stratert classified the formal information system according to the level of decision-making complexity of the system's users. These levels are defined to be: strategic decisions, tactical decisions, and technical decisions.

The level of strategic decisions is characterized by a great deal of data uncertainty. The information necessary to support this future oriented process includes information on those external factors which are unique to the particular user and his operational environment. This complexity required by strategic decisions differs dramatically from the complexity of tactical decisions which deals specifically with short-term goals. These goals have little regard for any strategic or long-range planning, concerned only with well-defined objectives and the planning and controlling of such operations. The last level, technical decisions, refers to controlling and monitoring the actual implementation of those well-defined tasks. This level ensures completion in the most efficient and effective manner.

Representations of information are, therefore, concrete materializations of information. Structured information, as suggested, is a framework of data for explicitly classified entities in the real world.
Uses of Information

Sanders (1983) stated that information is needed in virtually every field of human thought and action. This information is not only essential to individuals who use it to achieve personal ends, but also to those decision makers in organizations. These decision makers must perform certain basic tasks or functions in order to achieve organizational goals. The goals pursued differ, but the basic tasks generally involve acquiring, allocating, and controlling all resources for the success of the organization. How well these functions are carried out is dependent upon how well the information needs of managers are being met. Beyond the basic organizational information, the question of what information is needed can only be answered in broad general terms because individual administrators differ in the ways in which they view information, in their analytical approaches to it, and in their conceptual organization of relevant facts.

Porat (1976) has also shown that the number of people employed in information activities (researchers, scientists, technologists, teachers, planners, controllers, etc.) has increased enormously in the last century, from a few percent 100 years ago to over 50% in 1980.

Information technologies are actually penetrating man's activities and important examples of this penetration are evident. New technologies are being introduced extremely rapidly, not only because of their processing advantages, but also because the underlying logic
is simple and understandable.

Large businesses are using information technologies and robots principally for general control and administration systems. Yet small and medium size firms have found a means of making profitable use of these new technologies by exploiting the flexibility they offer.

System Verification

Verification of system correctness has become an important and complex aspect of the total system development process. Many technical methodologies have recently been developed to test a system's correctness leading to several formal proofs of such methods. The goal of this theoretical work has been to provide a systematic formal approach to a technology which has in the past been ad hoc and intuitive. A number of interesting theoretical results have been proven which give insight into the testing process but which cannot be considered practical program testing tools. The most widely studied formal validation method is the inductive assertion technique for proving the correctness of programs. The proofs of correctness approach to validation is not currently useful except for the validation of relatively small combinatorial algorithms.

Freeman (1975) contended that in the testing approach to system validation, the correctness of a system is proved over a finite set of data by system execution on that data. Data here is defined to be a "grouping" of data for the purpose of storing, limiting, and instructing system responses. It is considerably easier to test a
system in this manner than to prove its correctness over all cases.

Goodenough and Gerhart (1979) attempted to provide a theoretical foundation for system testing, characterizing the properties of a completely effective test selection strategy. A test selection strategy is completely effective if it is guaranteed to discover any error in a program. By using computability theory, Howden (1976) indicated that it is not possible to construct a testing strategy that is guaranteed to discover all errors in a program. Howden then identified three classifications of common errors found in any one system: computations, input, and subcase.

White and Cohen (1980) expanded on the definition of errors, introducing the class of domain errors defined to be the subset of a system's domain which causes a path to be followed incorrectly. These errors can be caused by incorrect predicates in branch statements or by incorrect computations that affect variables in branch statement predicates. White and Cohen described a set of constraints under which it is possible to reliably detail domain errors.

Hantler and King (1976) indicated proofs of correctness are indeed theoretical approaches to validation rather than empirical or pragmatic, and testing can be considered as the initial step in an inductive proof. The initial step of the proof requires the first cases to be proven prior to any occurrence of induction. In this sense, testing is closely related to and built on the foundation of correctness proving methodologies.

The diagram of Figure 1 depicts the traditional process of systems development in which system validation is the last phase in a
four-part process. The necessity of moving the validation phase to an earlier position in the development cycle has become increasingly apparent. The types of validation methods that can be integrated into the development process and used during particular phases of the process are closely related to the types of system documents generated during the phase.

![Diagram of Development Process]

Figure 1. Development Process.

Two general types of validation activities can be carried out during each of the three phases of the development process: static and dynamic analyses. Static analysis involves an analysis of system properties that can be derived directly from system documents and
without execution of the system. Dynamic analysis, however, requires execution of the system and a testing methodology based heavily on the development of a controlled finite test data set.

Static analysis attempts to demonstrate the truth of an allegation defined to be a weak theorem about the system. These allegations can be classified according to whether they address the structural, syntactic, semantic, interprocedural, or some other category of program feature. Fagen (1976) indicated one of the most commonly used practical methods for validation is simply that of rereading design documents and source code. If the allegation is true then the property that the allegation addresses is not present in the system; otherwise, there is something wrong with the system.

Dynamic analysis seeks to exercise a system in a controlled and systematic way to both demonstrate that required functions are present and that unwanted functions are absent. Such a situation creates the need for process elements relating to objectives of the analysis, data necessary to meet the stated objectives, results verifying (or not) the objectives, and measurement standards used to evaluate the results of the process. In the ideal, the set of objectives, data, results, and measurements about each test would be achieved in some way as backup information; the same situation could then be set up and run later for system verification as a result of a system change.

Huang (1975) introduced the basic notions of dynamic testing based on detailed path analysis in which full knowledge of the contents of the source program being tested is used during the testing process. A common test criterion is to have every statement in the
program executed at least once. Huang suggested that a better criterion is to require that every edge in the program digraph be executed at least once. This is equivalent to requiring each decision outcome in the program to be executed at least once.

Functional testing, according to Howden (1980), involves the selection of program tests that evaluate each of the functions implemented by a system. The usual practice is to identify functions from requirements specifications. Howden indicated the necessity of design functions, invented during system design, and requirements functions to be of major importance in this form of dynamic analysis.

Deutch (1982) introduced the "partition analysis method" as a composite of path testing, symbolic analysis, and consistency checking based on specification analysis. The basis of the method is the use of symbolic evaluation methods to partition the "input space" of the candidate system into subdomains in a way that assures that on each such subdomain, individual data elements are processed uniformly by both the implementation and the specification. This, in effect, divides the input domain into more manageable parts.

Summary

This research addresses a method of designing and implementing an ongoing data verification process. This process is applied to an existing information system which has many of the characteristics of those systems defined above.

Chapter II of this research examines other investigations of the problem of semantic analysis. Much of this research addresses
semantic analysis as it applies to compiler theory and artificial intelligence with little regard for real life information systems and the inherent need for consistent data.

Chapter III presents a formal definition of the finite state automaton. Along with the generalized deterministic automaton and the Moore Machine, a deterministic finite state automaton is also defined and its uniqueness explained. The concept of a balanced deterministic finite state automaton is then presented, drawing on the definitions of both the generalized deterministic automaton as well as the formal definition of the Moore Machine.

The actual implementation of balanced automaton as a data verification tool is described in Chapter IV. A brief history of substance abuse data collection efforts in Michigan defines the environment for which this system was developed. Incorporation of this data consistency system required close examination of those software engineering standards which existed. The standards currently accepted in the field are described and justification is given for those chosen for implementation. The operating environment of the existing data system is also described. The implementation process is detailed with the results of the completed system being reported.

Chapter V provides a summary of the work presented in this research from both a theoretical and practical standpoint. The list of recommendations for continued research suggest several avenues for formally proving the automaton of this research as well as practically applying various theoretical machines to practical situations.
CHAPTER II

RELATED RESEARCH

Given the importance placed on information in today's society, it is extremely important to collect the most accurate information available. The syntactic analysis (validating the correct value of the data element entered) of the data found in the information system described in this research is performed during data entry in an expeditious manner. The logic applied to this analysis (described in Chapter IV) is straightforward. The semantic analysis (verifying the correct relationships between data elements) of these data is not, however, trivial.

The literature reviewed here is a general survey of research related to data verification and the various applications of finite state automata (though it is certainly not an exhaustive review given the large amount of research available). Few studies were found which dealt with data verification performed after data entry. The nature of the database used in this study indicated a need for examination of alternatives to entry-actuated verification. The use of finite state automata to develop the logical relationships between data elements closely parallels several applications described in this chapter.
Review of Literature

Dippel and House (1969) indicated that there exists a strong need to verify the data correctness as they are collected. This verification process can be performed using a number of methods available for checking the accuracy of data collection and recording. The cost as well as the ease of error detection and correction is likely to affect the particular method of verification selected.

Visual verification of source documents can be used to determine if collected data are accurate, complete, correct in format, and reasonable in appearance. Once the data has proceeded beyond the entry stage of the processing system, errors become more difficult to detect and correct. Visual verification is not, however, very effective. It is slower, more costly, and less efficient in catching errors than other types of verification processes. Visual inspection is also very costly in time and money. It is also extremely difficult to verify the semantic correctness of the received data using visual inspection.

Stair (1981) indicated a need for a three step process for verifying data correctness. These steps are: validation, verification, and editing. The validation proposed by Stair coincides with the visual verification step defined by Dippel and House (1969). Stair believes this process will sufficiently check the syntactic responses found on the entry document.

Once the validation phase is complete, Stair (1981) indicated the second step to be data verification. This verification is proven
by establishing equipment or software to reread the original source and verify it against the stored value. This process verifies the entered information against the stored data. This verification ensures proper placement and acceptance of the necessary data by the system. It does not, however, assure the correctness of the information. Stair's final phase is editing. This takes place before the data is entered into the computer system for processing. Here, the data is checked to make sure that it is in a form that can be processed by the application programs.

Rademacher and Gibson (1983) indicated that there are two activities in data entry that the system designer includes in the system to reduce errors and thus increase the accuracy level. These activities are verifying data items before they are entered into the system and editing data items immediately as they enter the system.

The method of data verification proposed by Rademacher and Gibson (1983) closely parallels the definition used by Stair (1981) as listed above.

Rademacher and Gibson (1983) expanded the previously defined notion of editing to include several on-line tests conducted by the machine once the data has been input. These tests are: validity testing, completeness testing, and limits testing.

Validity testing will accept key data from a user and perform those tasks necessary to verify its existence in the database. Though a proper match does not guarantee that the data entered is 100% accurate, it greatly increases the probability that the key is correct. This test is, of course, invalid for new data.
The completeness test as defined by Rademacher and Gibson (1983) simply ensures that all fields have received a response from the user. If the data is not entered, either the record being entered would be rejected completely or the key would be rejected. In either case, the user would go through a re-entry process.

In the limits tests outlined by Rademacher and Gibson (1983), a test would be given to the input data, verifying its value against a base and bound value stored somewhere in the system. Should the data item fall outside this range, the edit program would send an error message to the user entering the data.

Meadow (1976) indicated that data should be tested for validity upon entering a computer. There is no uniform standard indicating how to test or how intensively to test; at the least, however, data values that might prevent subsequent programs from operating correctly or efficiently should be detected. Failure to carry out tests may also lower the quality of files by permitting erroneous data to be stored in them, and possibly retrieved and used before discovering the error. An example of the kind of item that should be tested is a field to be used for computing a storage address. An out-of-range value could produce miscalculation of an address from a key, resulting in a record being stored in an area reserved for another file or for programs. Thus, a data change could modify a program and cause a major system malfunction.

Meadow (1976) also alluded to the fact that this testing can vary in complexity. Simple testing could verify that all characters of a field are members of the correct alphabet or that a numeric
field has a value within a stated range. More complex tests may involve conditional expressions such as: IF RANK = 6 THEN 5500 < PAY < 10,000. Functions of fields such as RELEASE DATA = ENTRY DATE + 365 may also be applied to data verification.

The discovery of invalid fields calls for decisions regarding whether the entire record must be rejected, only the field rejected, or merely an annotation made on an error listing that will go to a data control clerk. Total rejection may be expensive and may introduce intolerable delays in entering information into the system. Ignoring the problem of validation is implicitly a decision to permit erroneous data to accumulate and be used.

Weber, Stucky, and Karszt (1982) described a classification scheme of integrity assertions. The implementation posed by Weber, Stucky, and Karszt indicates the use of a subsystem to evaluate semantic integrity at the completion of all transactions and prior to database updating. Use of this scheme requires either holding transactions in the machine for later evaluation, or storing such transactions on a secondary device until the transaction mode has been completed. The first solution would require vast amounts of main memory should the transaction process be a constant routine. The latter solution suggests the creation of a temporary file. This solution greatly increases the time factor by the input/output factor of the host machine.

Lundberg (1982) described a method of checking the correctness of an information model. By using predicate calculus, he defined the universe of discourse in the example below:
This universe of discourse implies the perceived abstract knowledge to be represented as follows:

\[
\begin{align*}
X(\text{SEC}(X) & \rightarrow \text{EMP}(X)) \\
X(\text{EMP}(X) & \rightarrow Y(\text{SAL}(Y) \text{ AND } \text{PAY}(X,Y)))
\end{align*}
\]

These well-formed formulas can be translated into English to read:

- All secretaries are employed.
- Every employee has a salary.

On examining the consistency of information systems, Lundberg (1982) continued this analysis using predicate calculus, admitting that the process, although a fundamental necessity, is difficult to verify.

The use of finite automata as a tool of understanding has been crucial to the successful ventures of language theory and the necessary compilers needed to drive these languages. Aho and Ullman (1977) defined a recognizer for a language L as a program that takes as input a string x and answers "yes" if x is a sentence of L and "no" otherwise. Clearly the part of a lexical analyzer that identifies the presence of a token on the language is a recognizer for the language defining that token. This recognizer is best represented by a finite automaton.

Sampson (1976) indicated that information processing can be carried out by machines (finite automata) built from rather simple components. A modular net is an example of the processing Sampson
indicated as an outcome based on finite automata. This net can best be explained as an electronic digital circuit, capable of producing output based on the sequence and arrangement of the input signals. This can be represented by an automata.

Sampson (1976) also showed how finite automata acts as an acceptor for regular expressions. By seeking a simple and formal means of describing a class of sequences, the resulting machine would have the capability of describing the nature of all classes of sequences for which an acceptor can be designed.

Hartmanis (1968) associated finite automata with the mechanical, constructive aspects of mathematics by defining automata as the study of abstract computing devices, the classification of these automata, and the study of their computational power.

Hopkins and Moss (1976) indicated a simple finite automaton (mechanical in nature) might test items from a production line for weight, size, conductivity, or some other quality, and represent a satisfactory item by an output of 1. A defective item would then be identified with an output of 0. Not only are defective items removed, if they occur too frequently, the production line is halted.
CHAPTER III

FINITE STATE AUTOMATON

Davis and Weyken (1983) defined a finite automaton as a very limited computing device, which, after reading a string of symbols on input, either accepts the input or rejects it, depending upon the state of the machine when it has completed the state lookup based on the input string. The machine begins by reading the leftmost symbol in the string, starting in a specified state called the initial state. If at a given time the machine is in a state of q_i reading a given symbol s_j from the input, the machine moves one symbol to the right and enters the state q_k. The current state of the automaton and the symbol from the input being examined completely determine the automaton's next state. An "epsilon move" in a finite automaton is defined to be a move from state q_i to q_j on a null input.

Hopcroft and Ullman (1979) defined the finite automaton as a mathematical model of a system with discrete inputs and outputs. The state of the system summarizes the information concerning past inputs that is needed to determine the behavior of the system on subsequent inputs. Formally, a finite automaton consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from an alphabet. For each input symbol there can be several transitions out of each state. One state is the initial state in which the automaton starts. Some states are designated as final or accepting states. Aho and Ullman (1977) defined a
deterministic finite automaton to be an automaton with no transitions on input epsilon, and for each state s on any input a, there is at most one path labeled a leaving s. Thus the finite automaton defined by Hopcroft and Ullman is nondeterministic. The deterministic automaton, when completed with the input string will be in state q_f and will either accept or reject such input. The nondeterministic machine, when completed with the input string, will be in a set of states \( \{q_1 \ldots q_n\} \) and will accept the input string if and only if the set \( \{q_1 \ldots q_n\} \) contains a state found in the set of final or accepting states. The finite automaton is formally denoted by a five-tuple \((Q, \Sigma, \delta, q_0, F)\) where \(Q\) is a finite set of states, \(\Sigma\) is a finite input alphabet, \(q_0\) is the initial state, \(F\) is a proper subset of \(Q\) containing all final or accepting states, and \(\delta\) is the transition function mapping \(Q \times \Sigma\) to \(Q\).

**FSA Representation**

A directed graph or transition diagram is associated with a finite automaton where the vertices of such a graph corresponds to the states of the automaton. If there is a transition from state q to state p with an input of a, there exists an arc from q to p labeled a in the transition diagram. A finite automaton is said to accept a string x if and only if the sequence of transitions corresponding to the symbol inputs from x results in a state found in the set of final states.

Aho and Ullman (1977) defined the use of finite state automaton as applied to lexical analysis. Through their discussion of applying
transitional diagrams to the concept of lexical analysis, Aho and Ullman clearly indicated how the use of deterministic finite automaton can be used to accurately evaluate a character string and return a token to the calling parser as determined by the designed grammar.

FSA Implementation

The use of finite automata dictates a need to store the control data as defined by its transition diagrams in such a way as to keep all transition rules readily available in main memory, allowing access based on the current state and the next input character \((q_i, k_j)\). This definition suggests implementation as the indices of a two dimensional array. Normal conventions use the rows of such an array as the states of the finite automaton and the column indices the representation of the current input. The coordinates of state \(s_i\) and input \(k_j\) yield a next state. This table lookup is then evaluated for termination. These conventions work well until \(\Sigma\), the input alphabet, becomes large and the transition diagrams are not completely specified. All unspecified coordinates contain some form of delimiter, usually zero. The array of size \(m \times n\) then becomes identified with a sparse matrix. There is no precise definition of when a matrix is sparse and when it is not, but is rather a concept that is intuitive. The accepted method of representing a sparse matrix is a three-tuple defined by \((I_s, I_i, T)\) where \(I_s\) is defined to be the state of the original array, \(I_i\) is the column index of the original array determined by the input character, and \(T\) is the transition state value found at the coordinates \((I_s, I_i)\) in the original array.
This representation is restricted such that $I_8 \times I_4 = T \leftrightarrow 0$ (or the delimiter used). Although this representation could be used for any table lookup, the software necessary for lookup represents a cost. The cost is, however, balanced when a sparse matrix is used, as the memory space saved represents a large savings.

One limitation of the finite automaton as defined is that its output is limited to a binary signal representing either acceptance or rejection. Models in which the output is chosen from some other alphabet have been considered. A machine with output associated with the current state is defined to be a Moore Machine.

The Moore Machine

A Moore Machine is represented by a six-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where $Q, \Sigma, \delta,$ and $q_0$ maintain identical definitions as those detailed in the formal definition of finite automaton. $\Delta$ is the output alphabet and $\lambda$ is a mapping from $Q$ to $\Delta$ giving the output associated with each state. The output of this machine $(M)$, in response to input $a_1 a_2 \ldots a_n$, $n \geq 0$, is $\lambda(q_0), \lambda(q_1) \ldots \lambda(q_n)$, where $q_0, q_1, \ldots, q_n$ represents the sequence of states such that $\delta(q_{i-1}, a_i) = q_i$ for $1 \leq i \leq n$. The deterministic finite automaton may be viewed as a special case of a Moore Machine where the output alphabet is $\{0,1\}$ and state $q_1$ is accepting if and only if $\lambda(q_1) = 1$.

The definition of the Moore Machine above will be used in this research to justify the use of finite automaton in determining data verification. The definition will be modified to show the transition to yield output if and only if $\lambda(q_4)$ is found in the set of states.
defined to be all possible error states. By applying this concept to a string character by character rather than to a string determined by end-of-string delimiters, a comprehensive and straightforward data analysis model may be constructed.

Balanced FSAs

The use of deterministic finite state automata for semantical data verification can best be defined as a searching technique for locating those data elements found in a particular data set which, when viewed as a whole, are inconsistent with their definition of a component of a logical data set. The deterministic nature of the balanced automaton closely parallels the use of balanced tree structures used in various searching techniques.

Horowitz and Sahni (1976) outlined the advantages of using balanced tree structures rather than optimum tree structures for searching, noting that if there are N nodes in the balanced tree, and N is of a high order, the resulting search creates a tree index that can be searched, making a very small number of disk accesses, even when the number of entries is quite large. This, of course, only holds true for those balanced tree structures sufficiently small enough to remain in main memory during processing. Horowitz and Sahni also indicated that the algorithm for insertion of data into a balanced tree structure is conceptually much simpler than the corresponding insertion algorithm necessary for those structures created under the concept of the optimum tree. The Adelson-Velskii-Landis tree structure, defined to be an optimum tree, was introduced in 1962...
to be a balanced tree with respect to the height of subtrees. As a result of the balanced nature of this type of tree, dynamic retrievals can be performed in a complexity of $O(\log N)$ time if the tree has $N$ nodes in it. The search time necessary for the balanced tree can be calculated as $c \log n + d$ where $c$ and $d$ are constants.

The concept of a balanced finite state automaton dictates that, regardless of the path taken, the number of nodes visited is consistent. Two conditions must be imposed on a general deterministic finite automaton in order to fulfill this requirement. The first condition dictates that regardless of the path taken in the automaton, the same number of nodes or states will be visited. This is the critical condition when a fixed length string is used as input. As a production tool, the finite automaton must be capable of dealing not only with fixed length input strings, but also fixed fields. These attributes are also necessary for a deterministic automaton that is used as an error detection device rather than a pass/fail device. This condition excludes the visitation of any error states in its requirements, as they are defined to be epsilon moves to a non-error-recovery state. This concept also supports the engineering design feature of conceptual simplicity when designing the transition modules. These modules can be constructed to use control files as delimiters rather than using the standard conventions of spaces or other specified delimiters.

The second condition imposed on a deterministic finite automaton to make it balanced requires modularization of each sequence of verification steps. The logical break for modularization coincides
with the natural break in the ADES data found in admission and discharge data sets. This can further be broken down to modularize by input forms, and still farther by the logical relationships of the data elements themselves. This concept can best be defined by Figure 2.

Figure 2. Testing Format.

Figure 2 depicts the entry and exit of FSA modules based on the logical ordering of data found on the ADES form sets. This concept enhances module development and provides adequate test points for system maintenance.

This new machine can best be defined by the seven-tuple \((Q, \Sigma, \Delta, \delta, \lambda, q_0, F)\) where \(Q\) is the finite set of states \((1 \ldots 85)\), \(\Sigma\) is the finite set of input \((A \ldots Z, 0 \ldots 9, \text{blank})\), \(\Delta\) is the output alphabet containing nineteen defined character strings, \(\delta\) is the transition function appearing as coded table traversals, \(\lambda\) is the coded action based on the current state, \(q_0\) is the initial
state (1), and \( F \) is the set of final states \( \{85\} \) which, when detected, will trigger a request for the next data set and will reinitialize the state traversal variable. This definition, although a composite of the definitions for finite automaton, Moore Machines, and balanced automaton, completely defines the device necessary for solution to the problem at hand and meets the criteria for a balanced automaton as defined above.

The modularization criteria required for a deterministic automaton creates enviable conditions for system maintenance and enhancement. Should a form be replaced, the submachine defined by the transition diagram could simply be removed without excessive rewriting of the system by indicating the start state to be the final state of the module. Likewise, should data definitions be revised requiring changes in the verification process, the internal module of any one finite automaton could easily be removed, replaced, or modified without rewriting any portion of the software.

These two conditions imposed on the deterministic finite automaton create an environment conducive to a healthy, manageable system. There are, however, drawbacks that must be considered. These conditions result in the creation of a deterministic finite automaton with more than the minimum number of states. This creates a need for not only additional space requirements, but also an increase in central processing time requirements.

The deterministic automaton of Figure 3 is an integral design element in the interform consistency system for ADES. This automaton in its present state is balanced, as movement through the diagram,
using any path (excluding epsilon moves) yields the same number of nodes visited.

Figure 3. Sample Balanced FSA

In an attempt to minimize this machine, it is clear that State 7 can be eliminated, as it makes absolutely no alternative move based on input. This immediately reduces the automaton by one state, giving the transition diagram in Figure 4.

Figure 4. Reduction of Balanced FSA.

Hopcroft and Ullman (1979) introduced a minimization algorithm for finding the minimum state deterministic finite automaton by letting \( \equiv \) be the equivalence relation on the set of states \( (M) \) such that \( p \equiv q \) if and only if, for each input set \( x \), the resulting move
from p with input x is an accepting state as well as the move from q
using input x is an accepting state.

In each step of this minimization process, (see Figure 5) incre­
mental separation of the original states occurs based on the defined
transition from each state based on the input character. This mini­
mization results in the deterministic automaton shown in Figure 6.

\[
\begin{align*}
\{1,2,3,4,5,6,7,8,9,10,11,12\} & \{13\} \\
\{1,2,3,4,6,8\} & \{5,9,12\} & \{10,11\} & \{13\} \\
\{1,6,8\} & \{2,3,4\} & \{5\} & \{9,12\} & \{10,11\} & \{13\} \\
\{1\} & \{6\} & \{8\} & \{2,3,4\} & \{5\} & \{9\} & \{12\} & \{10,11\} & \{13\}
\end{align*}
\]

Figure 5. Minimization Steps.

\[
\begin{align*}
1 & \rightarrow 2,3,4,7 & 6 & \rightarrow 10,11 & 13
\end{align*}
\]

Figure 6. Minimized FSA.

The FSA of Figure 6 indicates that movement from State 6 to
State 10,11 can occur in either one or two moves. This finite au­
tomaton of Figure 6 is, therefore, not balanced.
This indicates an isomorphism between those equivalence classes of $\equiv$ that contain a state reachable from $q_0$ by some input string and the states of the resulting minimal state set ($M'$). The steps of this minimization algorithm are detailed in Figure 4 as applied to the automaton for Figure 3. The input alphabet is defined to be the set of all characters, the numeric set 0 to 9, and epsilon.

Implementation of such an automaton unduly increases the complexity of the driver module, creating a subsystem not conducive to standard maintenance. Adding a "dumb" state to this minimized automaton would balance the movement and still reduce the original deterministic automaton by two states. However, the error definition capability of the original transition diagram is lost in the minimization process. This would indicate that the use of a deterministic finite automaton for error detection precedes the requirement for state minimization. The automaton of Figure 5 will detect errors; the type of error is, however, lost in the minimization process.
CHAPTER IV

APPLYING FSA TECHNIQUES TO ADES

The need for substance abuse services in Michigan has grown substantially over the past few years. Unfortunately, the funding available for these services has declined during the same period. This reduction of funding and increase in need has resulted in increased demands on fewer people, creating a situation where statistics become a major contributor to those decisions on how best to allocate resources. These statistics are used as justification for federal grants as well as resource allocation not only by the state, but by counties as well.

In order to properly assess the need of services in a region and project those needs into other areas, an evaluation of the existing client base is needed. A group of substance abuse professionals in Michigan devised a set of forms, each containing a series of questions designed to report social, demographic, substance abuse history, and any personal data of the client considered to be vital to identifying common problem areas. These forms were developed to ensure statistical capability and yet take into account real life situations. The forms found in Appendix A represent the set of questions presented during the admission of a client. The 551 form gathers control and general information concerning the client's presence, client description, social standing, and methods of payment. The 552 form of Appendix A is dedicated to those questions determined
by substance abuse professionals to indicate substance abuse problems.

The forms found in Appendix B are used during the discharge of a substance abuse client. This series of questions attempts to ascertain the success of the treatment extended to the client. Many of the questions are identical to those asked during admission, providing a base for statistical analysis of the success as it relates to those measures determined to indicate success as defined by society. Several questions are specific in evaluating the abuse of substances against those problems presented at admission. Both sets of questions have been defined as a "snapshot," presenting a before and after picture of the client.

The substantial increase in the need of substance abuse services has created a proliferation of data which can no longer be supported by conventional manual methods. The sheer volume of information has exceeded the capabilities of the remaining staff available for compilation of such data. Analysis of the data gathered was also extremely difficult due to the large quantity of data. The advent of an automated data system was readily apparent.

Working Environment

The ADES Data Collection System was designed in 1978 to alleviate many of the above problems. Since that time, the system has resided in the computing environment found at Western Michigan University in Kalamazoo, Michigan. This computer center takes advantage of a Digital DECsystem-1099, running the TOPS-10 v2.0 operating system.
The ADES users, located throughout Michigan, are connected to this computer system via telephone circuits using dial-up modems and hardcopy terminals to enter and retrieve information. These entry sites receive client data from those substance abuse programs in their region and enter these data into the ADES system with software designed for the input of such data.

The ADES system has been developed in ANSI COBOL 68. This decision, made in 1978, has allowed the system to grow while keeping the system portable for future expansion. The use of COBOL also allowed the use of ISAM files, best defined to be generalized file structures inherent to COBOL. The method of applying ISAM files to the ADES system closely parallels the definition of a relational database definition. Several files of manageable proportion contain data which, when examined together, represent all the information available for any particular client. This implementation method has allowed substantial growth while maintaining a relatively consistent level of performance. The design of the DECsystem-1099 has allowed for the construction of identical file structures for each user, maintaining their uniqueness through a disk allocation process. The control software for the system is also stored in a separate disk area and can be accessed by any of the ADES users. This access is restricted by design to the users' data only, providing a level of data security. There does, of course, exist the capability to access all user areas from a central control area for consortium-wide analysis.
Over the years the ADES Data System has expanded its usefulness. The ADES users may now use the system to plan for allocation of funds for a fiscal year, plan for provisions of service to their region, and plan the staff time of existing substance abuse programs. The system has also been expanded to include those accounting practices imposed on the regional agencies by the state. These enhancements have provided those reports previously produced manually as required by those governing bodies overseeing the disbursement of funds.

The environment in which the ADES system exists supports the Statistical Package for the Social Sciences (SPSS). The use of this package has been integrated into the ADES Data System. As such, the ADES users have become sophisticated in their use of SPSS, capable of evaluating client information from a myriad of standpoints, presenting the data to various governing bodies.

The Problem

The information found in the ADES Data System may be used as a foundation for decisions which could have far reaching effects in the field of substance abuse services in Michigan. The limited funding available coupled with the increased need for these services has led to many decisions made on services being offered and in what quantities. The ADES data have been used by state and federal agencies in making these and other important decisions.

The evaluation of this data for important decisions mandates the verification of consistent data. Should a significant amount of data contain inconsistencies of a similar nature, a conclusion could be
reached which would have detrimental effects on the field.

Recognition of this potential problem resulted in annual audits performed on the data found in the ADES Data System. This audit presented problem areas for evaluation and correction for the upcoming fiscal year. The audit did not, however, identify which data sets were inconsistent but rather identified those classes of inconsistencies which presented problems. Production of audit findings often took 3 months of valuable time; a time factor making data verification an almost impossible task. The turn around time of any data set being corrected usually ran in excess of 12 months, far too long to be useful in any timely analysis. A method for automating data consistency verification was needed with several methods being examined.

The edit function of the ADES Data System at input is a valuable tool for ensuring information gathered from the field falls within those parameters outlined and agreed to by the majority of substance abuse professionals. This function is accomplished by comparing each data element entered against base and bound values found in a file keyed by the form and question number identified. This system function evaluates individual elements only. This file has fields for the form identification number, question number, base value, and upper bound value, using the first two fields as a key for lookup. When a response is entered, the information system reads this file and compares the entered value against the base and bound figures found in the record. Should a numeric value be required for input and an alpha character is entered, TOPS-10 interrupts the program and
requests a numeric response from the user. This is the same process invoked when an alpha character is required and a numeric value is entered. The excess time required to read various files for verification of consistency between data elements requires a time factor which quickly removes this verification process from real time editing, as the computed response time to users is far in excess of any reasonable time frame.

The only tool currently available in the ADES Data System for evaluation of data consistency is SPSS. Based on the results of the committee's findings, over eighty-five such reports must be produced to evaluate the consistency of entered data. Once manually verified, other SPSS reports must be run to search the data base for identification of those clients whose records indicate an inconsistency. This process is cumbersome, time and computer expensive, and subject to misinterpretation by those field technicians producing such reports. The proliferation of data at all levels of substance abuse agencies would severely curtail any honest attempt by ADES users to undertake such a time consuming and complex task.

Software Engineering

Attempting a project of this magnitude and complexity requires standards to ensure proper design and construction. These standards, once implemented, ensure minimal elapsed time between start-up and completion. Aron (1974) indicated that the development process is viewed as an organized set of activities that produces a program system. Aron's statement exemplifies a strict organization which
also ensures maintainability. Assessing those standards in existence for possible implementation into the ADES organization led to a further examination of the software engineering field.

Cotterman et al. (1981) defined software engineering to include the systematic design and development of software products and the management of the software process. Shooman (1983) more generally defined software engineering to be the process of taking designs based on theory and ensuring that they are constructed to meet the imposed standards. Shooman also emphasized the importance of proper documentation, indicating the first milestone in any design process to be that of sound documentation. He detailed this idea to be the act of reducing the conceptual design to memorandum form, including diagrams, charts, drawings, series of applicable equations, and all necessary algorithms.

Documentation Standards

After examining those standards related to documentation typically found in any software engineering technique, the format of the suggested documentation was analyzed. Rubin (1979) indicated the System Requirements Study to be the principal formal documentation derived from the investigative study. This stand-alone document includes the purpose for the document through a proposed implementation plan. The study addresses the current system, identifying its weaknesses in relationship to the project needs. The study also addresses those management issues presented as a result of project implementation.
The System Requirements Study, with several modifications (see Appendix C), met the criteria defined in software engineering standards, and also met the needs of Areawide Data and Evaluation Services.

In some ways pseudo-code is more flexible than a flow chart. One can easily increase the level of detail in pseudo-code whenever more details are needed to prevent confusion. Another major advantage of pseudo-code is that it can be created, modified, controlled, and reproduced with ease by use of a terminal, text editor, a number-date convention, and a printer associated with a modern interactive computer system.

The relatively small size of the ADES organization allowed a great deal of flexibility in the selection of a design medium. It did, however, eliminate those design tools requiring excessive amounts of production time. The use of flow charts and the Hierarchy plus Input-Process Output (HIPO) technique in its purest form were two such documentation aids determined to be production excessive. After careful consideration it was determined that a process encompassing those engineering standards directly applicable to ADES were necessary, yet the mediums suggested were too cumbersome. The concept of Level I algorithms was therefore developed by ADES. These algorithms were designed to be written in a pseudo-code which closely parallels COBOL-81. This mild restriction was included to allow a smooth transition from design through the development phases and leading, finally, to implementation. The only other restriction on this pseudo-code called for indentation of code lines similar to the
structure imposed in the language PASCAL. This restriction not only allows for ease of analysis, but also quickly indicates undo complexity caused by the design imposed. By minimizing restrictions on this Level I pseudo-code, the design is better able to develop in a manner closely related to the methods imposed by prior systems; manual or automated in nature.

System Design

The system design process begins with a concept as understood from the System Requirements Study. In translating this concept into software, there exists a need to document such a process, giving the first opportunity to intuitively examine its complexity and correctness.

The second phase of system design requires a breakdown of the Level I algorithm into modules and detail of the process to be performed in each module. The formalization of this modularization of the algorithm closely parallels the use of Warnier Diagrams.

A Warnier Diagram can best be defined as an organizational chart laid on its side. However, other things can be done with Warnier Diagrams that are not possible with other hierarchical forms of organization—for example, dealing with logic, repetition, and sequencing. The use of the Warnier Diagram as a standard for ADES was rejected due to the complexity of the diagram and the excessive production time necessary. The modular definition table as defined in the ADES Programming Standards (See Appendix E) helps control module complexity and by its construction, provides the programming
documentation necessary for incorporation into the software.

Although the documentation aid HIPO was originally rejected due to the excessive production time required, the need to evaluate the design based on throughput remained. The shorthand notation defined by Rubin (1979) was adopted by ADES to meet this requirement. Short HIPO, when used in conjunction with decision tables, eliminates the need for the preparation of final documentation and produces specification documentation which is also suitable for use as final documentation. One of the most significant benefits of the use of Short HIPO is that the diagrams become a management aid for use in the control of the project work. Feedback, in the form of reviews of the Short HIPO diagrams, provides the information that management needs to monitor both progress and accuracy of the development effort. The value of this feedback is greater than when using customary documentation techniques because it is available before considerable effort is expended doing something in an incorrect manner. This is because the Short HIPO diagrams are available for review before implementation. When using other documentation techniques, the feedback information is not available until after the work is performed.

The approach approved by ADES for writing a structured program is to begin with a Level I (high level) program plan. Any loop repetitions and decisions which appear at this level are realized in the design by using IF then ELSE. Each major component of the design can be expanded in a similar manner. Such a technique is generally referred to as stepwise refinement. Shooman (1983) indicated that the most important advantage of structured programming is its
clarity. Much of the huge cost and difficulty of programming lies in
the complexity of the problems attempted and the chore of explaining
such ideas. A structured design clearly presents the control struc-
ture of the program and fits in well with top-down techniques.

The Solution

This project should allow each substance abuse agency using the
ADES system to receive monthly reports indicating those data records
containing inconsistent data. The memos generated by the system will
indicate by substance abuse program the client identification number,
the input form containing the erroneous data, the responses in ques-
tion, and the reason for the apparent inconsistency.

The criteria used in determining such inconsistencies followed
the Michigan Office of Substance Abuse Service's standards. These
standards define every data element and the interrelationships be-
tween these elements. The ADES Board of Directors shall also make
final approval of those data relationships which are inconsistent
with the intent of collection.

Verification of data consistency is a vital element for con-
sideration when a decision is made concerning target groups and how
to best serve Michiganders with substance abuse problems. The flexi-
bility of producing these reports over any time period allows verifi-
cation of either historical data to support past decisions or current
data to assist managers develop upcoming decisions.

The need to minimize CPU time was readily apparent, due to the
extensive input/output necessary to complete the search for client
data. Searching for these data was determined by realizing that a sequential read of the client-data-file must take place. This results in \( n \) reads where \( n \) is defined to be the total number of clients in the program. It was also realized that once the client passes the check for either admission or discharge date, there exists the possibility of up to four additional reads for each of these clients \( (m) \). Fortunately, the only output to the memo-file is the memo format (sixteen writes) and any errors encountered \( (p) \).

Therefore, the number of input/output operations can be formulated as:

\[
[n + 4m] + [p + 16]
\]

Analysis of historical data indicates that each agency averages 5,000 client admissions annually. From these 5,000 cases, approximately 3,000 clients are discharged annually, creating the final data set. The ADES system maintains all client data records on-line to ensure complete access by the substance abuse coordinating agencies. The ADES system is in its sixth year of operation. Analysis of data in past audits indicates at least one out of every five clients has at least one inconsistency.

These figures, coupled with the previously mentioned formula, indicate a total number of input/output access as follows:

\[
[30,000 + 2(2,000) + 4(3,000)] + [1,000 + 12(16)] = \alpha + 47,192
\]

This number quickly balloons when applied to all ADES users. The system currently services ten substance abuse agencies bringing the total figure to \( \alpha + 471,920 \) input/output access for production of a monthly report, where \( \alpha \) is defined to be those additional input/
output accesses necessary for clients with additional records.

This analysis follows the standard routines established in the ADES system. As such, an expedient method of verification was essential.

In determining the best data configuration for the finite state automata, the definition posed by Maurer (1977) of multi-dimensional arrays proved to present the best solution to this problem. Maurer defined a two dimensional array or matrix by indicating a node $k'$ is said to be a row successor of $k$ if $(k, k') \in \text{ROW}$ and $k'$ is a column successor of $k$ if $(k, k') \in \text{COL}$ where

$$\text{ROW} = \{k_{i,j}, k_{i,j+1} | 1 \leq i < m, 1 \leq j < n-1\}$$

and

$$\text{COL} = \{k_{i,j}, k_{i+1,j} | 1 \leq i < m-1, 1 \leq j < n\}$$

for an array consisting of $m$ by $n$ nodes.

Maurer (1977) defined row order by indicating a step size of $s$ of a rectangular array $B = (K, R)$ such that $k_{i,j} = k_{11} + s[n(i-1) + (j-1)]$. Thus the address of the node $k_{i,j}$, that is, the node in the $i$th row and the $j$th column can be computed directly from the address of the node in the first row and first column and the numbers $i$ and $j$.

The Interform Consistency System will examine all client information available. Should a client not have one or more of the four forms required for normal processing, the inconsistency evaluation performed by the system will not record any errors, but rather "drop through" that particular data set. This ensures all errors recorded are due to incorrect data collection or entry and not the result of
missing client data.

The method of verifying consistent data is table lookup. This process closely parallels the syntactical analysis performed in compilers, using lexical analysis techniques. This process should minimize the time required to examine data once it has been input to the software.

Once the ADES board decided to use a table driven routine, the task to define such a table was undertaken. Using the process Aho and Ullman (1977) described, the table was developed. The problems that arose indicated a need to establish software engineering standards for maintenance purposes. The final result was the balanced deterministic finite automata as previously defined. During the design of the balanced DFA, it was also decided to include start and exit states for each series of data verification steps. Although this expanded the size of the table, the maintenance benefits far outweighed any space restrictions. The modularization of the table imposed by this design should significantly reduce the time necessary for modification of the table, a task assured to be necessary over the course of time.

All client information will be read into the system and the elements to be analyzed will be moved into a string for use in table traversal. This process of creating a list of all elements to be examined circumvents the need for a complicated and cumbersome filtering algorithm.

Using the concept of a balanced automaton as defined by this research and the definition of finite automaton by Hopcroft and
Ullman (1979), the transition diagrams of Figures 7, 8, 9, 10, 11, 12, and 13 were created. These diagrams define a complete deterministic finite automata which closely parallels the definition of a Moore Machine, while preserving those restrictions as presented for a balanced automaton.

The output is written to disk taking full advantage of the "scratchpad" facility currently available on the existing computer system. The memos in this file are preceded by a page top flag for line printer output. The local program TPRINT will be used to produce on-line memos for distribution to the coordinating agencies. The line printer copy will be maintained at the ADES office for historical purposes.

Figure 7. Transition Diagram for Form 551.
Figure 8. Transition Diagram for Forms 551/552.

Figure 9. Transition Diagram for Form 552.
Figure 10. Transition Diagram for Form 651.

Figure 11. Transition Diagram for Forms 651/652.

Figure 12. Transition Diagram for Form 652.
Figure 13. Transition Diagram for Forms 552/652.

The generation of Interform Consistency Reports has become the responsibility of the ADES staff, thus freeing up more time for agency personnel to perform those tasks necessary for the day-to-day operation of their office.

Once the system was in operation, the coordinating agencies began to receive monthly reports delineating those data inconsistencies encountered on the previous month's input. The list of errors are then distributed to those affected substance abuse programs for correction. Once the errors are changed, the list is returned to the agency for input. The entry process takes advantage of various input options which allow an entry to be made for single data elements, thus minimizing the time necessary to effect these changes.

The Interform Consistency Reports have become a vital part of the audit process performed by the coordinating agencies when making a site visit to a substance abuse program. This process expedites the audit, saving the agency both time and aggravation.
Implementation

Once the System Requirements Study was reviewed by the ADES Board of Directors, approval of the project was given and a consistency committee established. The committee was composed of both technical systems professionals and substance abuse professionals from the field as well as the State Office of Substance Abuse Services.

The committee reviewed all data found in the ADES system and determined the list of related data as they appear on the form set required for all substance abuse clients. The committee not only indicated necessary consistency between data elements of a particular form, but also those data determined to require consistency between the forms of the client data set. The resulting list of consistency checks is found in Appendix D. This list was then presented to the ADES Board of Directors, who gave the approval necessary for project continuation. The completion of this list, coupled with the System Requirements Study, presented the foundation for continued design.

Development of the transition diagrams followed, using the list of desired consistency evaluations. The design of these diagrams followed those concepts of a balanced finite state automaton as presented in this research. During the development process of these diagrams, several problematic situations of data arose resulting in a need for further clarification of data definitions by the consistency committee. This process of diagram design was also instrumental in determining the method and format of client data which is located in
various files in the ADES system.

Once complete, these transition diagrams became the foundation of the system. Throughout the implementation process these diagrams would serve to either verify a particular process or correct the misnomer under which a particular module was being developed.

Following those programming standards as approved by ADES, concurrent work began on the modified HIPO and Warnier documentation. This documentation process helped to design a system with well-defined and reasonable routines having no unnecessary complexity. As indicated in those programming standards, all documentation is used during software development, incorporating all modular definitions in the software. This, coupled with level number prefixes for the software modules, creates a system well designed for maintenance.

The first module completed was the one identified as necessary to create the driver table. Various methods included: a static file containing the table, a separate program to generate a file containing the table, a subroutine in any convenient language to be called from the main system, and a simple assignment module in the main system. Due to space requirements, the first two methods were rejected. The decision to use a subroutine was rejected to minimize those conversion problems encountered during a system migration believed to be in the near future. Completion of this module was evaluated against those transition diagrams previously defined using the walkthrough process. The successful results of this evaluation were presented to the ADES Board of Directors.
Based on the advantages of the walkthrough evaluation as presented by Freedman and Weinberg (1982), the walkthrough process became an integral part of the system evaluation process. Freedman and Weinberg contended that the major advantage of a walkthrough is that it does not make many demands on the participants for preparation in advance. Where there are large numbers of participants, or where the participants come from diverse organizations not under the same operational control, it may prove impossible to get everyone prepared for the review. In such cases, the walkthrough may be the only reasonable way to ensure that all those present have actually looked at the material.

As delineated in the System Requirements Study, Phase II of the implementation process dealt with the development of the modules necessary to read input, finding consistency errors as they occur. Several software modules were developed in accordance with the programming standards to complete those tasks outlined in the HIPO documentation developed earlier. The module definitions were used to maintain a manageable level of complexity per module and assure that an adequate level of detail was maintained. Once these modules were finished, testing occurred using various black box and white box techniques defined in the System Requirements Study.

Meyers (1979) indicated that black box test cases exploring boundary conditions have a higher payoff than those cases that do not. Boundary conditions as defined here are those situations directly on, above, and below the edges of input equivalence classes. Boundary-value analysis requires that one or more elements be
selected such that each edge of the equivalence class is the subject of a test. Equivalence class as used here denotes those classes of input which represent either valid or invalid input to a system. Due to the benefits of boundary-value analysis, this method of system testing was incorporated into Phase II.

The other black box testing methodology employed in Phase II was that defined to be error guessing. It is difficult to give a procedure for this technique since it is largely an intuitive and ad hoc process. The basic idea is to enumerate a list of possible errors or error-prone situations and then produce test cases for each item on the list. This technique helped to remove those conditions which were not to be accepted. Again the use of the transition diagrams was instrumental in the design of these cases.

The white box testing techniques used on this system included statement coverage and decision/condition coverage. The statement coverage technique requires the tester to construct an input sequence such that every system statement is executed at least once. This technique ensures that statements perform as they were intended. The decision/condition coverage testing technique requires sufficient test cases such that each condition in a decision takes on all possible outcomes at least once, and each point of entry is invoked at least once.

These testing procedures, coupled with the black box tests described earlier, created a reasonable set of test conditions and, although not a complete test procedure, produced adequate assurance of system performance.
The final phase of implementation involved design of the system output format, production of the system modules to create the formatted output, and lastly, the method of output dissemination. Informal communication with system users and those managers of substance abuse administration lead to the final design of a memo format. The distribution method followed the one currently employed for other reports in the ADES system.

The testing of the completed system was relatively straightforward, relying on the incremental tests performed at the completion of each milestone. The final test was a system run on actual substance abuse data covering a 6-month period, previously examined manually in a fiscal data audit. The results were identical.

Implementation Results

The development of the Interform Consistency System held many interesting challenges. The system design required a decision process for determination of those logical relationships of data which were acceptable and coincided with those standards as they exist in the field of substance abuse services. The design also specified the use of various high level techniques in such a manner not previously identified as a direct application. The requirements imposed on such a system as related to maintenance and enhancement required development of programming standards not previously needed in the ADES environment. The method of disbursement of the system results forced several extra methods of production, leading to a comprehensive and organized manner of distribution.
Once the initial study was accepted, a committee of substance abuse professionals examined the data found in the ADES system and, working conjunctively with the Michigan Office of Substance Abuse Services, determined the logical relationships between the data elements. The list of these relationships is found in Appendix D. This list, along with the connotation of a balanced finite automata, resulted in those transition diagrams necessary to complete the task of data verification. These diagrams were verified with those static analysis tests as identified in the ADES Programming Standards (see Appendix E).

Once the transition diagrams were verified, development of Hierarchy Input Process Output and a form of Warnier documentation was undertaken. This critical phase of the development process resulted in several modifications to the initial study due to apparent problems surfacing as a result of these procedures. Completion of this documentation resulted in an efficient and straightforward implementation.

The first production run of the verified software produced vast amounts of output indicating an excessive and troublesome amount of data inconsistencies. Examination indicated the error lists to be judged correct when evaluated against those logical relationships as defined by those previously mentioned substance abuse professionals. The committee reconvened to discuss the systems outcome and re-evaluate the criteria for data relationships. The results of this work led to a redefinition of several data elements and their relationships by the Michigan Office of Substance Abuse Services. This
reevaluation process will continue on an annual basis.

The annual data audit performed on the information found in the ADES system has been improved through the development of this Interform Consistency System. Prior to the implementation of this system, the audit examined data consistency through the production of 357 statistical reports using the Statistical Package for the Social Sciences (SPSS). Incorporation of the Interform Consistency System into the annual data audit has replaced these previously required SPSS reports with a single pass run. This improvement in the audit process has resulted in a reduction of production time in excess of 30%, a substantial decrease in CPU time, and has removed any mis-interpretation of statistics as presented in SPSS.

This Interform Consistency System, due to its reasonable use of computer resources and its user friendly output, has become an integral part of the ADES system. The system is executed on a monthly basis and the output is distributed accordingly. Analysis of consistency errors before implementation of this system indicates an overall improvement of an 80% decrease in inconsistency errors as reported.

The enthusiasm of those users for this system has resulted in a better defined environment for data collection. The dissemination of the system's output has substantially increased the awareness of those substance abuse professionals who are charged with collecting this data.
CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

The main focus of this research is the application of theoretical machines to the real world problem of data consistency verification. This semantic analysis is crucial to any information system with the potential of impacting major decisions for any field related to the data. This research introduces the concept of a balanced deterministic finite state automaton as a seven-tuple machine, a special case of a Moore Machine. The inclusion of the finite set of final states to this input driven machine makes the use of an automaton a realistic approach to the problem of semantic analysis. This finite automaton as defined by this research assumes the input to be consistent in length of string size. Each input is also assumed to be consistent by data element field. These assumptions substantially simplify the code necessary to traverse the state table which defines the balanced deterministic finite state automaton.

This research also deals with the inclusion of software engineering standards and techniques in the Interform Consistency Project. This need is based on the desire to apply this system to a dynamic data system where data definitions, locations, and collection processes are constantly undergoing revision. The development of the ADES Programming Standards was a direct result of the Interform Consistency Project. The standards were designed to include various features of several documentation techniques, creating an easily

53
understood and useful document identified as the System Design Note-
book. This document will become the foundation of any further work
to the Interform Consistency Project due to enhancements, bugs en-
countered, or revisions resulting from a dynamically changing re-
quirement. The coding section imposes on the system development only
those standards deemed essential to promoting stepwise, top-down
structured programming. Although relatively restrictive in the basic
requirements, these standards are quite flexible in design, allowing
creativity to become an integral factor in the design process. The
ADES Programming Standards also allow for maintenance procedures as
generally accepted practice in the field of software engineering.

Recommendations for Further Research

There are a number of areas related to this research where
additional investigation would further the efforts put forth here.

This research introduces the balanced deterministic finite state
automaton as a viable mechanism for semantic analysis of data found
in an information system. The seven-tuple machine identified as the
balanced deterministic finite state automaton must undergo rigorous
methods of formal theorem proving as it relates to those machines
currently identified as standards in the study of automaton. This
formal proof would allow theoretical application of this machine in
computer science.

This research applies the balanced automaton to an actual appli-
cation, resulting in a set of transition diagrams which define a
state table with dimensions of 85 by 29. This relatively small,
nonsparse matrix works well in its environment, virtually eliminating any time complexity due to system memory paging. Should the dimensions necessary for a large application exceed this paging, are there any representations which would better represent the automaton? If the table can be minimized to meet this paging restriction, could the process be generalized and applied to any balanced automaton? If the minimization proves successful, how will the complexity of the driving software be affected and does this violate the software engineering standards established for maintainability?

The assumptions of this research create an environment conducive to a deterministic finite state automaton. Do these assumptions exclude the development of a nondeterministic finite state automaton for use in semantic analysis? Should such a machine be developed, how would the software engineering standards concerning complexity be affected? Would such a machine substantially reduce the space requirements of a system?

This research applies balanced finite state automaton theory to a specific data system in a specific environment. To what class of problems does this method apply? Could this process be applied to a data system with an unknown length of input with varying fields?

The balanced finite state automaton, by its definition, is not always an optimum solution to a problem. This definition has several conditions which could be applied to artificial intelligence inference systems resulting in system generated transition diagrams and the associated state table. Would this automated method unnecessarily increase the size of the table? If so, how would minimization
affect the outcome?

In conclusion, this research contributes to the body of research focused on practical applications of theoretical machines to data consistency verification. The concern of important decisions made based on inconsistent data is certainly a significant aspect of this problem.
Appendix A

ADES Admission Forms
ADES ADMISSIONS FORM

Name: ____________________________________________ Next of Kin: to be notified in case of emergency
Street: _____________________________________________
City, State: ____________________________________________ Phone: ____________________________
Phone: ____________________________

Place of Employment: _______________________________________

ADES ADMISSIONS FORM

DOCUMENT TYPE: [ ] Original [ ] Correction [ ] Deletion

Program ID: ____________________________

Service Category
1. Emergency Medical Services (EMS) (130)
2. Appraisal Service Program (ASP) (330)
3. Other
4. Other

Entrance Assessment
6. General Assessment
7. Emergency Assistance Program (EAP)
8. Other Community-Based (CMAB)

Client ID: ____________________________

Admission Date (Mo, Day, Year)

10. Staff ID: ____________________________

Treatment Service Description
a. Appraisal Service Program (ASP/130)

Other
b. General Assessment
c. Emergency Assistance Program (EAP)
d. Other Community-Based (CMAB)

Service Date: ____________________________

13. Client ID: ____________________________

From Other Areas
a. Self
b. Court referral offensives
14. Treatment Service Description
c. Department of Corrections
d. Court referral - other

Treatment Service Description
15. Treatment Service Description
e. Parole
f. Secretary of State's office

16. County/City/Neighborhood Code (CVC)

g. Mental health program
h. School Systems

17. Client Date of Birth (Mo, Day, Year)

i. Other human service program
j. Other community services

18. Primary reason for treatment referral

a. Minor
b. Major
c. Other

19. Admission Type

a. First admission
b. Re-admission

20. Source of referral

From a Substance Abuse Program

21. Client's sex
a. Male
b. Female

From Other Areas
22. Race/Ethnicity
a. Black
b. White
c. Hispanic
d. Native American

23. Current marital status
a. Married or Cohabiting
b. Never Married

c. Divorced
d. Separated

24. Veteran status
a. Yes
b. No

c. Single or Never Married

25. Highest formal school grade completed

26. Current employment status

a. Employed Full-Time (35) or more FTE
b. Employed Part-Time (Fewer than 35) FTE

c. Unemployed, not in labor force (e.g., homemaker, retired, not looking for work, etc.)

27. Current annual income

28. Major sources of financial support for the last 30 days

a. Job
b. Social Security
c. Unemployment
d. Pension

e. Welfare
f. Child Support

29. Health insurance coverage

a. Blue Cross/Blue Shield
b. Metropolitan Life Insurance Co.
c. Family

d. Other

30. Number of months in present job


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ADCS ADMISSIONS FORM

Program ID: 
Service Category: 
Client: 

DOCUMENT TYPE: □ Original □ Correction/Update □ Deletion

31. Present legal status
   a. No pending court cases
   b. Pretrial
   c. In jail
   d. Other legal status
   e. Other

32. Number of Arrests and Convictions
   a. Possession of drugs
      A - Possession of drugs
      B - Sale or distribution
      C - Traffic QUEL-QUEL
      D - Traffic OWI
      E - Other traffic violations
      F - Crimes against property
      G - Crimes against person
   g. Other

33. Client perception of present health
   a. Excellent
   b. Good
   c. Poor

34. CLIENT SUBSTANCE USE INFORMATION (Use Drug Codes for this item)

<table>
<thead>
<tr>
<th>Drug (Use Code)</th>
<th>No of Days Used in Last 30 Days</th>
<th>Year of First Use (Last 2 Dates)</th>
<th>Is Drug Prescribed (Yes or No)</th>
<th>Longest Use (in Days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Primary</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2) Secondary</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

35. Client Substance Use Goal
   a. Abstinence
   b. Controlled use

36. Other problem areas (circle up to three major areas)
   a. Education/school
   b. Occupation
   c. Financial
   d. Social relationships
   e. Legal

37. Medical F.D. =

COMPLETE QUESTIONS BELOW FOR DRUNK DRIVING ASSESSMENTS ONLY

38. Blood Alcohol Content
   □ 1

39. Length of Assessment
   1. 16 minutes or less
   2. 16 - 29 minutes
   3. 30 - 44 minutes
   4. 45 - 59 minutes
   5. 60 - 89 minutes
   6. 90 - 119 minutes
   7. 2 - 4 hours
   8. 4 hours or more

40. Assessment recommendations
   1. Education
   2. Interven
   3. Outpatient
   4. Residential
   5. Education and treatment
   6. No services recommended
   7. Mental Health Assessment
   8. Other

41. Charge for Assessment
   1. No charge
   2. Less than $50
   3. $50 - $79.99
   4. $80 - $299.99
   5. $300 - $499.99
   6. $500 - $799.99
   7. $800 - $999.99
   8. $1000 and over

42. Payment of Costs
   1. Full payment
   2. Full payment: time payment
   3. Partial
   4. No payment
   5. No charge
   6. Unknown

43. Court Response
   1. Agree
   2. Disagree: No referral
   3. Disagree: Different referral
   4. None
   5. Other
   6. Unknown

44. Client Response
   1. No referral recommended
   2. No referral recommended by court
   3. Client admitted to alcoholism program
   4. Client admitted to treatment
   5. Client admitted to education and treatment
   6. Client on waiting list
   7. Client did not follow through
   8. Unknown

45. CODED REMARKS: □ 1 □ 2 □ 3 □ 4 □ 5 □ 7 □ 8 □ 9 □ 10 □ 11 □ 12 □ 13 □ 14 □ 15 □ 16 □ 17 □ 18 □ 19 □ 20 □ 21 □ 22 □ 23 □ 24 □ 25 □ 26 □ 27 □ 28 □ 29 □ 30.

NOTES
Appendix B

ADES Discharge Forms
<table>
<thead>
<tr>
<th>DOCUMENT TYPE:</th>
<th>☐ Original</th>
<th>☐ Correction</th>
<th>☐ Deletion</th>
</tr>
</thead>
<tbody>
<tr>
<td>10. Program ID:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. Program Type:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. Client ID:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Discharge Date (Mo. Day, Yr):</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14. Start:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. County-Village/Township Code:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. Reason for discharge:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17. Transfer/Referral To:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. Discharge type:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19. Current marital status:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20. Highest formal school grade completed:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21. School/skill program completed during treatment?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22. Current employment status:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23. Current annual income:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24. What was your major source of financial support since admission?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25. Number of months in present job:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
# ADES DISCHARGE FORM

### DOCUMENT TYPE:
- [ ] Original
- [ ] Correction/Update
- [ ] Deletion

### 26. Method of Payment
- [ ] Self Pay
- [ ] Blue Cross/Blue Shield
- [ ] Metropolitan Life Insurance Co.
- [ ] Aetna Life Insurance Co.
- [ ] Delta Dental Plan
- [ ] Edible Life Insurance Society of U.S.
- [ ] Prudential Insurance Company
- [ ] Revere Insurance Company

### 27. Present legal status
- [ ] No pending court cases
- [ ] Probation
- [ ] Parole
- [ ] In jail

### 28. NUMBER OF ARRESTS AND CONVICTIONS SINCE ADMISSION

<table>
<thead>
<tr>
<th>Drug (use Codes)</th>
<th>No. of Times Arrested</th>
<th>No. of Times Convicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Possession of drugs/alcohol</td>
<td>1 2 3 4 5+</td>
<td>0 1 2 3 4 5+</td>
</tr>
<tr>
<td>B: Sale of drugs</td>
<td>0 1 2 3 4 5+</td>
<td>0 1 2 3 4 5+</td>
</tr>
<tr>
<td>C: Out/PAT</td>
<td>0 1 2 3 4 5+</td>
<td>0 1 2 3 4 5+</td>
</tr>
<tr>
<td>D: Crimes against property</td>
<td>0 1 2 3 4 5+</td>
<td>0 1 2 3 4 5+</td>
</tr>
<tr>
<td>E: Crimes against persons</td>
<td>0 1 2 3 4 5+</td>
<td>0 1 2 3 4 5+</td>
</tr>
<tr>
<td>F: Other</td>
<td>0 1 2 3 4 5+</td>
<td>0 1 2 3 4 5+</td>
</tr>
</tbody>
</table>

### 29. Client's perception of present health
- [ ] Excellent
- [ ] Good
- [ ] Poor

### 30. CLIENT SUBSTANCE USE INFORMATION (USE DRUG CODES FOR THIS ITEM)

<table>
<thead>
<tr>
<th>Drug (use Codes)</th>
<th>No. of Days Used in Last 30 Days or Since Admission</th>
<th>In Drug Preceding (Yes or No)</th>
<th>Length of Non-Use in Last 100 Days or Since Admission (in Days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Primary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2: Secondary</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 32. OTHER PROBLEM AREAS (USE PROBLEM CODES FOR THIS ITEM)

<table>
<thead>
<tr>
<th>Problem Code</th>
<th>Question</th>
<th>Score</th>
<th>Score</th>
<th>Improved</th>
<th>Eliminated</th>
<th>New Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Primary</td>
<td>a b c d e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2: Secondary</td>
<td>a b c d e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3: Tertiary</td>
<td>a b c d e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 33. CODED REMARKS

<table>
<thead>
<tr>
<th>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30</th>
</tr>
</thead>
</table>

Notes
Appendix C

System Requirements Study
A SYSTEM REQUIREMENT STUDY for INTERFORM CONSISTENCY REPORTING

Written by:
Michael L. Richardson
Executive Director
A.D.E.S., Inc.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose of Document</td>
<td>3</td>
</tr>
<tr>
<td>Current System Description</td>
<td>3</td>
</tr>
<tr>
<td>Problems with the Current System</td>
<td>4</td>
</tr>
<tr>
<td>System Capabilities</td>
<td>4</td>
</tr>
<tr>
<td>Management Issues</td>
<td>4</td>
</tr>
<tr>
<td>Conceptual Design</td>
<td>5</td>
</tr>
<tr>
<td>System Constraints</td>
<td>6</td>
</tr>
<tr>
<td>Implementation</td>
<td>6</td>
</tr>
<tr>
<td>Appendix A</td>
<td>8</td>
</tr>
<tr>
<td>Appendix B</td>
<td>10</td>
</tr>
<tr>
<td>Appendix C</td>
<td>11</td>
</tr>
<tr>
<td>Appendix D</td>
<td>12</td>
</tr>
</tbody>
</table>
Purpose of Document

The Areawide Data & Evaluation Services, Inc. (A.D.E.S.) data collection system has been in operation since 1978. During its existence, the system has been used by substance abuse agencies of Michigan to compile vast amounts of data on the clients receiving services, and has attempted to provide statistical analysis of this. Analysis of this data has become crucial to substance abuse professionals in defining the target groups of services rendered as well as indicating where and how state funds are being expended for these services. The importance of this data requires that each data set gathered for clients be consistent; both from an inner data set standpoint, and by definition as defined by the State Office of Substance Abuse Services. This System Requirement Study attempts to define a subsystem which will meet these requirements. This subsystem will interact with the existing A.D.E.S. database, using all necessary data files and routines as defined in the A.D.E.S. Systems Documentation Manual.

Current System Description

The input mode of the A.D.E.S. system assures users that data input will fall within base and bound parameters established by question definition and form design. These edit checks are made independently, making no reference to previous or upcoming entries. These checks are easily incorporated, and are not difficult to modify. The response time associated with these edits falls well within the parameters of efficient operation.

The A.D.E.S. system currently provides output to substance abuse agencies in two modes: internal report production and reports generated using the software known as the Statistical Package for the Social Sciences (SPSS). The internal reports produce management information, enabling agencies to monitor their substance abuse programs, ensuring certain levels of performance based on client throughput, cost per unit of service provided, compliance with predetermined objectives, and adherence to all laws pertaining to substance abuse services in the state of Michigan. The reports available through SPSS enable A.D.E.S. users to determine statistics based on their requests. These reports may contain up to three variables to delimit the statistics while restricting record selection by up to seventeen conditional fields. With judicious use, these reports help identify trends and types of clients, allowing compilation of any standard statistic for all client data found in the system.
Problems with the Current System

The edit function of the system at input is a valuable tool for ensuring information gathered falls within those parameters outlined and agreed to by the majority of substance abuse professionals in the field. The complexity of checking for consistency of the data input requires a time element that quickly removes it from real-time editing, as the computed response time to users is far in excess of any reasonable time frame.

The only possible means of evaluating data consistency currently lies in the judicious use of SPSS. Adequate evaluation requires eighty-five such reports per client. Once run, these reports must then be "proofed" for either verification of correctness or sent to the applicable substance abuse agency for clarification. This is obviously an extremely inefficient process.

Compounding this problem is the need for time to produce such analytical output. The proliferation of data at all levels of substance abuse agencies would severely curtail any honest attempt by A.D.E.S. users to invoke such a process. An automated system charged with evaluating data consistency and reporting in a clear, concise method is the only logical solution.

System Capabilities

When the objectives of this study are met, each substance abuse agency using the A.D.E.S. system will receive monthly reports indicating those data records containing inconsistent data. The memo generated by the system will indicate by substance abuse program the client identification number, the input form containing the erroneous data, the responses in question, and the reason for the apparent inconsistency.

The criteria used to determine such inconsistencies will follow the Michigan Office of Substance Abuse Services' standards. These standards define every data element and how each relates to the other. The A.D.E.S. Board of Directors shall also make final approval of those data relationships which are inconsistent with the intent of collection.

The generation of these reports shall become the responsibility of the A.D.E.S. staff, thus freeing up more time for agency personnel to perform those tasks necessary for the day-to-day operation of their office.

Management Issues

Once the system in enacted, the Coordinating Agencies will receive monthly reports delineating those data inconsistencies encountered on the previous month's input. The list of errors will then be distributed to those affected substance abuse programs for correction. Once the errors
have been changed, the list is returned to the agency for input. The entry process will take advantage of various input options which allow an entry to be made for single data elements, thus minimizing the time necessary to effect these changes.

The Interform Consistency Reports may then become a vital part of the audit process the Coordinating Agencies perform when making a sight visit to a substance abuse program. This process should expedite the audit, saving the agency both time and aggravation.

Verification of data consistency is a vital element for consideration when a decision is made concerning target groups and how to best serve Michiganders with substance abuse problems. The flexibility of producing these reports over any time period allows verification of either historical data to support past decisions or current data to help managers develop upcoming decisions.

Conceptual Design

This system is designed to interface with the A.D.E.S. data collection system. ISAM files containing client information are related to other ISAM files containing client specific data, including pointers for direct access to the information files. The latter files contain keys which allow for direct lookup for each client. For further file relationships, please consult the A.D.E.S. System Documentation Manual.

Because of these file relationships, minimal input is necessary, as a client is either rejected or passed through for analysis based on admission and discharge dates found after the first direct read. Should a client fall within the parameters established, input from various files will be done if, and only if, the client specific data file contains a pointer to the particular information file. Thus, each client specific data record is read once, and the potential exists for up to four additional reads per client.

The Interform Consistency Project will examine all client information available. Should a client not have one or more of the four forms required for normal processing, the inconsistency evaluation performed by the system will not record any errors, but rather "drop through" that particular data set. This ensures all errors recorded are due to incorrect data collection or entry and not the result of missing client data.

The method or verifying consistent data will be table-lookup. This process will closely parallel the syntactical analysis performed in compilers, using lexical analysis techniques. This process should minimize the time required to examine data once it has been input to the software.

All client information will be read into the system and the elements to be analyzed will be moved into a string for use in table traversal. This process of creating a list of all elements to be examined circumvents the need for a complicated and cumbersome filtering algorithm.
The state table to be used will be a deterministic, finite automata. Proven techniques will be used to minimize the table once initial design is complete. All incoming elements will be translated to an index based on definitions devised. This information will become part of the System Documentation Notebook (SDN). This document will contain all information necessary to understand, troubleshoot, and update the system. This requirement study shall also become part of the SDN.

The output shall be written to disk taking full advantage of the "scratch-pad" facility currently available on the existing computer system. The memos in this file are preceded by a page-top flag for line printer output. The local program TPRINT will be used to produce on-line memos for distribution to the Coordinating Agencies. The line printer copy will be maintained at the A.D.E.S. office for historical purposes.

System Constraints

The project, due to the need to interface with the A.D.E.S. database, several internally developed routines, and the computer system at Western Michigan University (W.M.U.), must be developed on a DECSystem-1099 using TOPS-10 in COBOL-68. All A.D.E.S.' data is stored in user defined ISAM files. This system has been designed to interface with this database system. Several macro routines from the W.M.U. Business College are also used to expedite data access, particularly across user boundaries. This project will fall under the A.D.E.S. system. As such, control routines must be accessed for record-keeping at various run stages. These routines, along with the entire A.D.E.S. system, have been designed in COBOL-68. For this reason, and the fact that COBOL-68 and COBOL-74 are incompatible, this project shall therefore be written in COBOL-68.

The interform consistency checks made by this system are designed for the current set of admission and discharge forms only. All historic data stored in old formats will not be analyzed in this system. Should this process be necessary, the routines delineated in the A.D.E.S. System Documentation Manual will need to be enacted to temporarily translate all data to the current format. The follow-up forms used in the A.D.E.S. system (set of three) are not analyzed due to the methods attempted in data gathering, and the fact that the Michigan Office of Substance Abuse Services currently defines follow-up to be a project (i.e., subject to change).

Development of this system shall be completed in accordance with the Programming Standards developed.

Implementation

The A.D.E.S. fiscal year 1983/84 work plan allocates time for this project from January through May, 1984. Due to unforeseen work plan modifications, it is determined that the project will not be implemented until September, 1984.
The development of the project will be implemented in three phases:

Phase 1 - develop a driver table to determine the logical relationships of the data.

Phase 2 - develop an efficient means of gathering data from each C.A. area and ensure collection in the proper format.

Phase 3 - develop the report producing software, ensuring all reporting is automated.

At the completion of each phase, a thorough test package will be developed and implemented. This package will not only test a particular phase, but any preceding ones. This follows the concept of incremental testing.

Each package will include, but not be limited to the below list of tests:

**Black Box**
- Boundary - Value Analysis
- Error Guessing

**White Box**
- Statement Coverage
- Decision/Condition Coverage

The completion of a phase represents a completed milestone. The completed incremented testing of phases to date represents yet another milestone. This results in the below list of milestones which, when reached, will be reported against.

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Completion of Phase 1</td>
</tr>
<tr>
<td></td>
<td>Test of Phase 1</td>
</tr>
<tr>
<td>2</td>
<td>Completion of Phase 2</td>
</tr>
<tr>
<td></td>
<td>Test of Phase 2</td>
</tr>
<tr>
<td>3</td>
<td>Test of Phase 1 and Phase 2</td>
</tr>
<tr>
<td>4</td>
<td>Completion of Phase 3</td>
</tr>
<tr>
<td></td>
<td>Test of Phase 3</td>
</tr>
<tr>
<td>5</td>
<td>Test of completed project</td>
</tr>
<tr>
<td>6</td>
<td>Follow-up report</td>
</tr>
</tbody>
</table>

Each milestone will be concluded with a report indicating actual time spent, problem areas, and all steps taken during the project's development.
INTERFORM CONSISTENCY LEVEL 1 ALGORITHM

Determine search conditions (dates, disk areas desired)

Loop thru disk areas (C.A.) until out
  Display and write memo-header
  Loop thru list of substance abuse programs
  Sequentially read client data files
    Loop thru form-data-files
      If form is available read into array
    End form-data-file-loop
  Test data using state table
  Display and write errors found
End program loop

Display and write memo-footing

End C.A. loop

APPENDIX B

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
APPENDIX C

START

FORM 551

FORM 551 to 552

START/END

NO 552 FORM

FORM 552 to 651

START/END

FORM 552

FORM 551 to 552

START/END

FORM 651

FORM 651 to 652

START/END

FORM 652

START/END

ADM TO DIS

END STA'1

START/END

FORM 652 to 651

START/END

TESTING FORMAT
FROM: Michael L. Richardson
TO: [C.A. Name]

Subj: Interform consistency check to date

1. The below table shows all interform inconsistencies for your agency by program.

<table>
<thead>
<tr>
<th>PID</th>
<th>CLIENT ID</th>
<th>FORM</th>
<th>INCONSISTENCY</th>
<th>CLASS OF ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>88888888</td>
<td>552</td>
<td>18a inconsistent with 15</td>
<td>Inconsistent Service</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>15 inconsistent with 11</td>
<td></td>
</tr>
</tbody>
</table>
Appendix D

Inconsistency Check List
SECTION I - INTERFORM CHECKS

A. 551

1. PID (10) and service category (11).
2. Service category (11) and service description (15).
3. Service category (11) and primary reason (18A).
4. Service description (15) and primary reason (18A).
5. Year of admission (13-3) and year of birth (17-3).
6. Admission type (18B) and number of programs (19).
7. Employment status (26) and major source (28).
8. Employment status (26) and number of months (30).

B. 552

1. Primary drug (341-1) and secondary drug (342-1).
2. Number of days used (341-2) and longest non-use (341-5).
3. Number of days used (342-2) and longest non-use (342-5).
4. Primary drug (341-1) by days used (341-2).
5. Primary drug (341-1) by longest non-use (341-5).
6. Secondary drug (342-1) by days used (342-2).
7. Secondary drug (342-1) by longest non-use (342-5).
8. Primary drug (341-1) by prescription status (341-4).
10. Primary drug (341-1) by year of first use (341-3).
11. Secondary drug (342-1) by year of first use (342-3).
12. Primary drug (341-1) by substance use goal (35).

C. 651

1. PID (10) by service category (11).
2. Reason for discharge (16) by discharge type (18).
3. Employment status (22) by major source (24).
4. Employment status (27) by number of months (25).

D. 652

1. Primary drug (301-1) and secondary drug (302-1).
2. Primary drug (301-1) and prescription (301-3).
3. Secondary drug (302-1) and prescription (302-3).
4. Days used (301-2) and length of treatment (CLI-10).
5. Days used (302-2) and length of treatment (CLI-10).
6. Longest non-use (301-4) and length of treatment or 180, whichever comes first.
7. Longest non-use (302-4) and length of treatment of 180, whichever comes first.
8. Number of days used (301-2) plus longest non-use (301-4) and length of treatment or 180, whichever comes first.
9. Number of days used (302-2) plus longest non-use (302-4) and length of treatment or 180, whichever comes first.
D. 652 (cont')

10. Primary drug (301-1) and days used (301-2).
11. Primary drug (301-1) and longest non-use (301-4).
12. Secondary drug (302-1) and days used (302-2).
13. Secondary drug (302-1) and longest non-use (302-4).

SECTION II - FORM TO FORM

A. 551 to 552

1. Service category (11) by primary drug (341-1).
2. Treatment service (15) by primary drug (341-1).
3. Primary reason (18A) by primary drug (341-1).
4. Primary reason (18A) by client substance use goal (35).
5. Client year of birth (17-3) by year of first use (341-3).
6. Client year of birth (17-3) by year of first use (342-3).
7. Year of admission (13-3) by year of first use (341-3).
8. Year of admission (13-3) by year of first use (342-3).

B. 651 to 652

1. Service category (11) by primary drug (301-1).
2. Length of treatment information.

C. 551 to 651

1. Grade at admission (25) by grade at discharge (20).

D. 551 to 652

1. Treatment service (15) by primary drug (301-1).
2. Primary reason (18A) by primary drug (301-1).

E. 552 to 652

1. Primary (341) by primary (301).
2. Possible secondary (342) by secondary (302).
Appendix E

Programming Standards
<table>
<thead>
<tr>
<th>SECTION I</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCUMENTATION</td>
<td></td>
</tr>
<tr>
<td>User Requests</td>
<td>2</td>
</tr>
<tr>
<td>Routine Functions</td>
<td>3</td>
</tr>
<tr>
<td>Project Proposals</td>
<td>4</td>
</tr>
<tr>
<td>Program Enhancements</td>
<td>7</td>
</tr>
<tr>
<td>Program Revisions</td>
<td>8</td>
</tr>
<tr>
<td>Program Bugs</td>
<td>9</td>
</tr>
<tr>
<td>Software</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SECTION II</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAMMING</td>
<td></td>
</tr>
<tr>
<td>Development</td>
<td>11</td>
</tr>
<tr>
<td>Program Structure</td>
<td>12</td>
</tr>
<tr>
<td>Trouble Shooting</td>
<td>13</td>
</tr>
<tr>
<td>Software Testing</td>
<td>14</td>
</tr>
<tr>
<td>Paragraph Labeling</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SECTION III</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDIX</td>
<td></td>
</tr>
<tr>
<td>Verb Set</td>
<td>A</td>
</tr>
<tr>
<td>User Request Form</td>
<td>B</td>
</tr>
<tr>
<td>Software Revision Form</td>
<td>C</td>
</tr>
<tr>
<td>HIPO</td>
<td>D</td>
</tr>
<tr>
<td>Program Logic Flow</td>
<td>E</td>
</tr>
</tbody>
</table>
These requests received by the ADES office are usually in the form of a phone call. They can, however, be in the form of a memorandum or a face-to-face request.

The ADES staff member accepting the request must complete a user request form (appendix B), completing the information requesting the name of the user making the request and the date, indicating themselves as the contact person. This last step will ensure end users of consistency in dealing with the ADES office. The section of the form identified as "Nature of Request" must also be completed by the contact person.

Once the pertinent information has been completed, the request, unless very unusual in nature, goes to the Programmer for completion. If the request appears unreasonable, it should be channelled to the Director for determination.

Upon completion of the work, a brief description of what was done is entered on the form, along with entering the completion date. The form is then returned to the ADES staff member who is acting as the contact person. This staff member is responsible for contacting the user and informing them of the work completed. Whomever does the actual work is responsible for entering this work in the daily programming journal.

Finally, once the Director initials the form, it is filed in the Programmer's file for that particular user.
ROUTINE FUNCTIONS

For all routine functions (backup, report productions, etc.) which are enacted on the entire data base, a file will be kept on each activity. In these files shall be kept all hard copies of the actual runs of each function. In the event the work is performed on a video terminal, a log shall be kept in the file allowing for entries indicating the work performed.

For all routine functions enacted on specific user areas, the same methodology is used on the files unique to these users.

All activities performed on the data system are recorded in the daily journal. This is essential to the verification process for particular problems. This daily journal must include all activity, system crashes, and any other unusual activity noticed on the system.

When tapes are involved (backup, data restoration), the program TAPES must be run to update the inventory delineating tape content. For further information on this program, please consult the System Documentation Manual.
A project proposal must be initiated with a System Requirements Study. This document, subject to board approval, clearly details the proposal, making comparisons with the existing system and the suggested project. The report contains the below listed items.

***SYSTEM REQUIREMENTS STUDY***

**Purpose of Document**
Describes why the current system needs revision or upgrading.

**Current System Description**
Describes the current situation; i.e., the capabilities of the current system. Types of output, selective reporting, audit trails, trend reporting, and available reports are all aspects of existing capabilities.

**Problems with Current System**
Indicates those weaknesses currently existing in the system. Timeliness, validity, usefulness, and cost are all indicators of problems.

**Alternative Design**
Describes various approaches to alleviating these problems.

**System Requirements**
Indicates the specific requirements of the proposed project. This is data specific in nature.

**System Capabilities**
States the projected capabilities of the proposal indicating those features which directly improve those problems listed above.

**Management Issues**
Describes critical issues that will affect the operational effectiveness of the system.

**Conceptual Design**
Describes the critical aspects of the system and presents an overview of the proposed system. NOTE: Be sure to describe all major processing steps.

**Implementation Outline**
Details an outline of activities necessary to complete the project. Indicates all milestones, providing time frames and reporting periods.
Once the System Requirements Study has been completed and accepted by the board, a SYSTEM DESIGN NOTEBOOK (SDN) identified by the project name will be started. The System Requirements Study will become part of the SDN. The SDN will contain the various components listed below.

System Flow
Shows in flowchart form, supplemented by narrative as needed, the general design of the overall flow of information from various sources, transformation of data into machine-readable sub-system processing, updating of data base, output reports, and distribution of output to final destination.

Functional Hierarchy
Uses a Visual Table of Contents (VTOC) to show in chart form the hierarchy of the system functions to be performed. This chart shall be limited to major user processing functions and should break down the system functions into major sub-processes within program modules. Please refer to the section on HIPO of this manual for criteria of the VTOC.

Detail Design
Indicates through the use of program logic flow charts, the calling procedures of paragraphs necessary for implementation. Section III of this manual delineates program logic flow charting criteria.

Data Relationships
Using HIPO, indicates data relationships, selection criteria, and the input, processing, output expectations of the system. A data dictionary should be developed as an appendix to the SDN in accordance with criteria found in Section III of this manual.

Input
Provides specific format of all input data desired. NOTE: Make sure to include screen layouts for all interactive input.

Output
Provides detailed explanation of all elements of the output and ensures that all reports are included in the format anticipated. Use of dummy data is quite useful in clearly stating the report's intent.

Control Logic
This should indicate any special logic developed for this project. The format is free, depending on the specific needs of the design.

Test Plan
This should be a detailed follow-up to that section of the System Requirements Study. It shall include test periods in relationship to milestones, the tests to be given, and the expected results of each test.
Should the project involve several pieces of software, a program summary section should also be included in the SDN and indicate:

**Program Number**
The sequential number assigned in the runstream.

**Program ID**
A full description of the program name.

**Input**
Identifies those data items used for input to this program and the format needed.

**Output**
Identifies those data items resulting from the work performed by this software.

**Program Summary**
A brief encapsulation of the program purpose within the sequence, including the conditions under which it is executed, and principal functions.

**Special Logic/Interfaces**
Describes exception logic, if any, or complex coding found in the program. Indicates interfaces with other programs in the system.

**Reports Produced**
Indicates whether this program produces any reports. A copy of any report produced should be attached.
SOFTWARE ENHANCEMENTS

Software enhancements can best be defined by those changes to software which do not disrupt the current processes of the system. A change to software which requires an effective date for all users is defined to be a revision and shall follow the documentation guidelines set forth in these standards.

More efficient techniques in computation, searching, or input/output are examples of enhancements. Report reformatting and basic content reconfiguration are also defined to be enhancements. These enhancements are oftentimes direct results of user requests or out-growths of special needs of various users.

Once a needs assessment has been done and the enhancement has been validated, work begins by first filling out appendix A. This form clearly outlines the work to be done, and uses a checklist to assist in assuring proper implementation.

The software should be documented in accordance with the guidelines set forth in these standards.

Upon completion of the implementation, the form is filed in the SDN (System Documentation Notebook) under the section detailing the specific software.
Program revisions are defined to be any work done to existing software which either restructures the entire design, or requires creation of new code for an additional requirement placed on the system. A revision, by nature, dictates much more time and energy than a program enhancement.

Documentation for a revision begins by completing a mini project proposal, including all those pertinent items necessary to justify the revision. This document should validate revising the existing code in lieu of creating new software.

Once the proposal's requirements study has been completed and approved, work begins by first filling out appendix C. This form clearly outlines the work to be done and uses a checklist to assist in assuring proper implementation.

The software should be documented in accordance with the guidelines set forth in these standards.

Upon completion of the revision, the form is filed in the SDN (System Documentation Notebook) under the section detailing the specific software.
SOFTWARE BUGS

Software bugs are defined to be deviations in the program based on the system's capabilities as outlined in the System Documentation Notebook (SDN). These deviations are not only deficiencies in meeting stated capabilities, but also those additional items not found in the final statement of capability. Any deviation from the expected process and output is considered to be a "bug".

Bugs are often discovered after a piece of software has been placed into production. Most indicators of software bugs come from the users. Occasionally, a bug is discovered during an enhancement or revision of another section of the software.

Once the problem has been identified through testing and analysis, work begins by first filling out appendix C. The severity of the problem will dictate the level of testing and should be so noted on the form. This form clearly outlines the work to be done and uses a checklist to assist in assuring proper implementation.

The software should be documented in accordance with the guidelines set forth in these standards.

Upon completion of the repair, the form is filed in the SDN under the section detailing the specific software.
SOFTWARE

In addition to external documentation (blackbox), there must be some form of internal (whitebox) documentation in order to minimize analysis time in the event of a revision, enhancement, or bug. The following list of standards is divided into sections defined in ANSI COBOL.

IDENTIFICATION DIVISION

PROGRAM ID Contains the name of the program as desired for identification to the user.

Remarks This section should contain text briefly indicating the origin of the software, the reason for its creation, date of implementation, number of revisions, enhancements, or bugs encountered, and the design and implementation team.

ENVIRONMENT DIVISION

CONFIGURATION SECTION

Source Computer Indicate machine, operating system and compiler version under which the software was compiled.

Object Computer Indicate, if different from source, the machine and operating system under which the software is used.

WORKING - STORAGE SECTION Variables should be grouped in relationship to their areas used. As area for general variables should also be reserved. Any array should be defined as to its usage.

PROCEDURE DIVISION

Each paragraph enclosed with starred lines and containing an exit shall contain sufficient documentation to adequately define the purpose of the paragraph. If necessary, design features will also be defined.

Whenever additional code is added due to enhancements or bugs discovered, the code added or commented on shall be separated from the other code by blank lines and clearly defined by a remark. When a piece of software undergoes revision, these separated code sections shall be completely integrated into the existing code.
The development process will conform to top down development standards. The below steps should be followed when any new software is developed.

- Ensure that program goals are clearly defined and well documented according to the standards found in this manual.

- All input/output data must be documented using an abbreviated HIPO. Appendix D contains all guidelines necessary for this development.

- Once all input/output has been developed, a flow chart shall be developed containing logical elements only. This flow chart is similar to a level I algorithm defined to be the outer language as observed by a user.

- The next step is to develop a program logic flow chart. This chart depicts each paragraph set (the paragraph name and its associated exit) and the logical relationships designed into the software. Appendix E is an example of such a chart.

- Code is then written. This code shall conform to those standards found in this manual.

- Implementation of the software involves adequate testing with proper documentation. Testing shall be done in accordance with the appropriate section of this manual.

The entire development process must, of course, follow sound documentation practices. Please refer to those applicable areas in the documentation section of this manual.
PROGRAM STRUCTURE

Once all documentation has been completed, the coding of the software shall be structured by the levels defined in the program logic flow chart. All level 1 paragraphs, in chronological order, shall appear as the first section of the procedure division. The second section shall contain all paragraphs designated as level 2, with all other levels following this process. Each section shall be separated by a row of stars, a remark indicating the section, and a final row of stars.

Every paragraph shall contain an exiting paragraph with a similar name.

EXAMPLE

0101-Display-STARTUP-MENU.
0101-Display-STARTUP-MENU-EXIT.

The exception to this rule is defined to be any one-line paragraph needed. This paragraph shall be incorporated into the calling paragraph.

EXAMPLE

0210-WRITE-OUTPUT-BITS.

... Perform MOVE-BIT varying BIT-counter from 1 by 1 until BIT-counter > 10.
...

MOVE-BIT.
  MOVE INPUT (BIT-counter) to OUTPUT (BIT-counter).
...

MOVE-NEXT.
...

0210-WRITE-OUTPUT-BITS-EXIT.
EXIT.

The use of GO TO statements shall be restricted to inter-paragraph routines and error/exiting calls only. The use of GO TO's which transfer control from one procedure to another will not be allowed.

The programming library shall be used whenever possible. The use of these copy statements shall be preceded by a remark indicating the code being copied over.

EXAMPLE

*File description:STAFF-file
Copy FD02

(}

-12-
TROUBLE SHOOTING

All paragraphs shall contain entry and exit statements to check the value of a flag. This flag shall be set to the level of the program logic flow chart, and all paragraphs in any level preceding or equal to the flag shall display all information necessary. This list of data shall contain:

1 - a message indicating paragraph name
2 - all variables and their values
3 - all arrays currently being manipulated
4 - all data being examined

The flag will be initialized to 0, thus not allowing any troubleshooting data to be displayed.

The use of arrays for verification, temporarily holding data, or making computations shall have all indicies base and bound verified before implementation. This check shall, upon determination of an error, display all necessary information and transfer control to an error paragraph where correct exiting will take place.

When troubleshooting software written before implementation of these standards, COBOL DDT will be used whenever possible. Should DDT not be available, a flag shall be created and conditional displays incorporated into the code. This will eventually bring all existing software into compliance with these standards.

When the need to troubleshoot arises, the documentation standards found in this manual shall be followed.
SOFTWARE TESTING

Each piece of software, when modified, will go through three levels of testing.

WALKTHROUGH
This will be the first test given the software. It may be done simply for one section of a program, or for an entire system. Method of implementation may also range from desk-checking to a team acting as a machine.

DATA-DRIVEN
This input/output system examines the correctness of the system from the outside. The data set should include base and bound conditions, different types of input, and erroneous data.

WHITE-BOX
This final test (which may be incorporated in the data-driven test) shall drive the software using data that will test internal base and bounds and enter as many branches as possible.

The data collected for the data driven and white-box tests shall be gathered and placed in a "Test Package". This package may then be used on the software when it undergoes enhancement. The Test Package should be modified when testing a software revision.

When an entire software package is designed, incremental testing shall be implemented. The System Requirements Study, as defined in these standards, shall contain milestones for evaluation and acceptance. These milestones shall coincide with the test performed on the developed software to date - thus ensuring incremental testing.

Remember - "Testing is the process of executing a program with the intent of finding errors."
PARAGRAPH LABELING

Initial Paragraph

Labels shall contain the various components listed below:

- **Four digits** - used to indicate program logic flow location with the first two indicating the logical level, and the last two indicating the number of the paragraph in the level.
- **Verb** - used to define action performed by the paragraph
- **Subject** - data being used by paragraph.
- **Condition** - identifies group of subjects (i.e., all, SPLIT, etc.)

Paragraphs internal to a routine shall contain the components listed below:

- **Verb** - used to define action performed by the paragraph
- **Subject** - data being used by paragraph
- **Reason** - reason for this paragraph

**EXAMPLE**

0101-LOAD-PROGRAM-IDS.
0201-READ-PROGRAM-FILE.
INITIALIZE-PID-TABLE-LOOP.
0101-LOAD-PROGRAM-IDS-EXIT.
VERB SET

WRITE - write data to file
SELECT - determine criteria for record selection
ACCEPT - interact with user for data
VERIFY - check input - either interactive or data read
PRINT - write output to file
LIST - to create lists for output or compilation for work
GET - extract data from a record currently in MM
COMPUTE - any mathematical function performed on data
DISPLAY - computer output to TTY
BEGIN - used to initialize section variables
FINISH - "wrap up" a section's work before exiting
INITIALIZE - used to initialize all variables for the startup of a program

Appendix A
## User Request

<table>
<thead>
<tr>
<th>CA:</th>
<th>Contact Person:</th>
<th>Request Date:</th>
<th>ADES</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Contact:</th>
<th>Completion Date:</th>
<th>Entered in Journal:</th>
<th>Date User Notified:</th>
<th>Directors Approval:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Nature of Request:**

**Work Done:**

---

**Appendix B**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
<table>
<thead>
<tr>
<th>SOFTWARE:</th>
<th>Revision</th>
<th>Enhancement</th>
<th>Bug</th>
</tr>
</thead>
</table>

Reason for work:

Input/Output affected by work:

### CHECK LIST

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>Copy source into 77341</td>
</tr>
<tr>
<td>2)</td>
<td>Protect (257)</td>
</tr>
<tr>
<td>3)</td>
<td>Make appropriate changes</td>
</tr>
<tr>
<td>4)</td>
<td>Document changes in code</td>
</tr>
<tr>
<td>5)</td>
<td>Run appropriate test package</td>
</tr>
<tr>
<td>6)</td>
<td>Compare original test results with new test</td>
</tr>
<tr>
<td>7)</td>
<td>Walkthrough test</td>
</tr>
<tr>
<td>8)</td>
<td>Whitebox test</td>
</tr>
</tbody>
</table>

Additional Comments:

(Appendix C)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
HIPO is a documentation technique which produces documentation suitable for use both as specifications and as the final documentation of a system or program. This abbreviated form of HIPO is designed to minimize the time spent in preparing the documentation. This system uses decision tables because they are easily understood by non-data processors, they can be proven to be correct and complete, and they permit rapid and ready identification of common processing functions.

The HIPO documentation defined here is composed of five segments:

1- a Visual Table of Contents (VTC)
2- an Overview Diagram for each "non-bottom level" function
3- a Detail Diagram for each "bottom-level" function
4- an extended description for each Overview Diagram and Detail Diagram requiring clarification
5- a Flow Chart for each Overview Diagram

VTC
The Visual Table of Contents resembles an organization chart. Each block corresponds to one function required to accomplish the ends of the system or program design. Each block is to be shown in its position in the hierarchy. This will allow the block to be connected to the one block at the immediately higher level to which it is a component, and is in turn connected to each of the blocks at the immediate lower level, which are components of it. Each block shall contain the name of the function it represents, and a unique identifying number serves as a page number, identifying the Overview Diagram or Detail Diagram which describes the function. Any of these blocks with subordinate blocks are identified as non-bottoming. Blocks without subordinates are referred to as bottom-level functions.

Overview Diagram
An Overview Diagram is required for each non-bottom block. Its purpose is to name the inputs, the lower-level functions, and the outputs associated with a function. This is accomplished by the use of three rectangular boxes, each of which contain a columnar list. The first box contains a list of inputs, the middle box contains a list of all subordinate functions, with the last box containing the output from the non-bottoming function.

Detail Diagram
This diagram is similar to the Overview Diagram except that it is used for bottom-level functions. This results in specific tasks being found in the center box rather than subordinate functions.
Extended Description

This description is used only to supplement and clarify Overview and Detail Diagrams. An example would be to list or define certain codes identified in the diagrams. This description should **not** reiterate what the diagrams indicate, but should only clarify when necessary.

Flow Chart

Flow Charts shall be used to indicate data flow for all Overview Diagrams created for the system or program. This Flow Chart shall follow the structured format, allowing only for constructs: Sequence, If-then-else, Case and Do-while. This system does **not** allow any decision making process to enter into the Flow Chart. This reinforces the definition of HIPO documentation.
Program Logic Flow

Prior to actual coding, a Program Logic Flow Table will be developed (see attached sample). All initial paragraphs will be at level 01 with all paragraphs being performed from that level being 02 in level. This process shall continue throughout the development of the software. Should a paragraph be called several times, the table will indicate where it is called each time, retaining the original paragraph identification number.

This process shall ensure that:

- no level \( n \) directly performs a paragraph of \( n + 2 \) or higher level;
- no level \( n \) performs any paragraph of \( n - 1 \) or lower level.

Appendix E
<table>
<thead>
<tr>
<th>Levels</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101-LOAD</td>
<td>0201-Determine</td>
</tr>
<tr>
<td></td>
<td>0202-Classify</td>
</tr>
<tr>
<td>0102-Menu</td>
<td>0203-Sell</td>
</tr>
<tr>
<td></td>
<td>0204-Sel12</td>
</tr>
<tr>
<td>0103-Pgm-EXIT</td>
<td>0205-Sel13</td>
</tr>
<tr>
<td></td>
<td>0301-Sort</td>
</tr>
<tr>
<td></td>
<td>Define each paragraph (only once)</td>
</tr>
</tbody>
</table>
Appendix F

Modular Definitions
This control paragraph calls other routines to initialize the state table and all variables. The main menu is also housed in this paragraph. Should a "blank" return be sensed, the system is exited from here.

This routine initializes all variables for the current run.

This paragraph, although exceptionally long, is quite straightforward. Each element of the state table is loaded as necessary. Where possible, one line internal paragraphs are looped through to reduce the coding lines necessary. The code is grouped by row of the state table, thus providing easy troubleshooting.

This paragraph is called when a response of "ADMISSION" is input to the command "INCON COMMAND: ". This routine will prompt the user for admission dates, used to indicate a range. Each date entered will be verified correct and a final edit check is made to ensure the correct chronological order of the dates. Should a date be entered in error, a message is displayed indicating what the error was and prompts again for the input. This routine is required for operation.

This paragraph is called when a response of "DISCHARGE" is input to the command "INCON COMMAND: ". This routine will prompt the user for discharge dates used to indicate a range. Each date entered will be verified correct and a final edit check is made to ensure the correct chronological order of the dates. Should a date be entered in error, a message is displayed indicating what the error was and prompts again for the input. This routine is required for operation.

This paragraph is invoked when a response of "LIST" is input to the command "INCON COMMAND: ". This routine will display the state table as it is currently being used. This routine is not required for normal operation.

This routine is invoked when a response of "PPN" is input to the command "INCON COMMAND: ". The code will allow up to ten 5-digit programmer numbers between the values 77316 and 77336. An input of blank will exit this routine. This routine is not required for normal runs, as the system will default to the list of all current valid programmer numbers.
This paragraph is called when a response of "DOCUMENTATION" is input to the command "INCON COMMAND;". The routine simply displays text indicating the use of the system, how it performs its functions, the distribution of the output, and where to refer to for programming specifics. This routine is not required for normal runs.

This paragraph is called when a response of "HELP" is input to the command "INCON COMMAND;". A list of all available commands is displayed with a short explanation of the function performed for each command. This routine is not required for normal runs.

This routine is the second control level for the system. It is invoked by a response of "INCONSISTENCY" to the command "INCON COMMAND;". The first section of this routine initializes all report variables which are generic to each output report to be generated. The paragraph then prompts for a decision on selecting all programs from each agency. This value will be used in the call to paragraph 0301-SELECT-PROGRAMS. The PPN-TABLE is then loaded based on the option of PPN selection. Two macro system internal routines (TTYWID & FILEPP) are invoked here, the first setting the necessary terminal width and the latter gaining access to the disk area assigned to the given PPN. This control routine will loop through the entire list of programmer numbers before relinquishing control to the next level.

This routine will first obtain the agency name and place it in the proper output location. The sub-paragraph SELECT-OPTION will allow for program selection based on the value of ALL-PGMS set in paragraph 0208. This sub-paragraph will only allow four responses: yes, indicating a desire to select programs; no, indicating all programs are valid; help, requesting further information; and, "blank", indicating a return to the previous level of control. To select programs, the routine 0401-SELECT-ALL-PROGRAMS is called, and the help request invokes 0402-DPY-PGM-SEL-HELP. When all programs are valid, the sub-paragraph ALL-PROGRAM-READ is actuated, placing all program data in the appropriate array.

This paragraph is called to allow the user to select those agency programs desired. The routine sequentially reads the PROGRAM-DATA-FILE, displaying the applicable information and requires a response of "YES" to enable the program for the report.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0402-DPY-PGM-SEL-HELP (COND)</td>
<td>This paragraph will display text explaining the commands available at this level of logic.</td>
</tr>
<tr>
<td>0302-WRITE-MEMO-HEADER</td>
<td>This routine will display the time and agency that is currently being examined. This information is vital to determine runtime efficiency and expected run duration. The sub-paragraph WRITE-MEMO will write the memo header previously loaded to the output file.</td>
</tr>
<tr>
<td>0303-GET-DATA-BY-PROGRAM</td>
<td>This paragraph is the third major control level, initializing those variables necessary whenever a new program data set is required. Control is passed on to the next level for dealing with specific client data. When the list of program is exhausted, the output memo record is completed, based on the quantity of errors encountered.</td>
</tr>
<tr>
<td>0403-GET-CLIENT-DATA</td>
<td>This routine is called from 0303 to loop through the client data files, verify the program ID, and pass control to the next level of code. This section of code will write a blank line in the output memo every time a new client ID is encountered if and only if there was output from the previous client examined. This paragraph is also used to verify that the current client data meets the criteria set previously. If the data passes, the client form data is then loaded and control is passed to the next level.</td>
</tr>
<tr>
<td>0501-SEARCH-LOAD-551</td>
<td>This routine is called to first determine if the 551 form exists for a client, and if so, load it into the test array. Once a form is found, a flag is set indicating its presence for use in a deeper level. A call is made to another paragraph to extract the form data, placing it in working storage. The sub-paragraph LOAD-551-DATA is invoked should the form exist. This routine will move the appropriate data into the test array in accordance with the state tables outlined in the System Documentation Manual.</td>
</tr>
<tr>
<td>0601-CHECK-FOR-ALL-FORMS</td>
<td>This routine loops through the CLIENT-DATA-FILE searching for a pointer to the form being examined. A value of 0 is returned if the client does not have the form.</td>
</tr>
<tr>
<td>0602-READ-OUT-FORM</td>
<td>This paragraph is used to enter the appropriate form data file and compute the beginning and ending locations of the form. The internal paragraph COMPUTE-PULLS is a standard routine in the ADES system, used in conjunction with the FORM-ID-FILE which details each form.</td>
</tr>
</tbody>
</table>
.0701-READ-ERROR-DUMP (COND)

This routine is used to display the key for the FORM-DATA-FILE and all pertinent client data when an invalid read is done on the FORM-DATA-FILE. The system is exited here for data analysis and system restart.

.0502-SEARCH-LOAD-552

This routine is called to first determine if the 552 form exists for a client, and if so, load it into the test array. Once a form is found, a flag is set indicating its presence for use in a deeper level. A call is made to another paragraph to extract the form data, placing it in working storage. The sub-paragraph LOAD-552-DATA is invoked should the form exist. This routine will move the appropriate data into the test array in accordance with the state tables outlined in the System Documentation Manual.

.0601-CHECK-FOR-ALL-FORMS

See above definition.

.0602-READ-OUT-FORM

See above definition.

.0701-READ-ERROR-DUMP (COND)

See above definition.

.0503-SEARCH-LOAD-651

This routine is called to first determine if the 651 form exists for a client, and if so, load it into the test array. Once a form is found, a flag is set indicating its presence for use in a deeper level. A call is made to another paragraph to extract the form data, placing it in working storage. The sub-paragraph LOAD-651-DATA is invoked should the form exist. This routine will move the appropriate data into the test array in accordance with the state tables outlined in the System Documentation Manual.

.0601-CHECK-FOR-ALL-FORMS

See above definition.

.0602-READ-OUT-FORM

See above definition.

.0701-READ-ERROR-DUMP (COND)

See above definition.

.0504-SEARCH-LOAD-652

This routine is called to first determine if the 652 form exists for a client, and if so, load it into the test array. Once a form is found, a flag is set indicating its presence for use in a deeper level. A call is made to another paragraph to extract the form data, placing it in working storage. The sub-paragraph LOAD-652-DATA
is invoked should the form exist. This routine will move the appropriate data into the test array in accordance with the state tables outlined in the System Documentation Manual.

.0601-CHECK-FOR-ALL-FORMS
See above definition.

.0602-READ-OUT-FORM
See above definition.

.0701-READERROR-DUMP (COND)
See above definition.

.0505-STATE-LOOKUP
This paragraph is called and contains the major work involved in this logical level. The routine initializes index variables and enters into a sub-loop to bounce through the state table. When a character is read, it is checked to determine if it can stand alone, or whether it is but part of a response. Once this has been verified, it is then converted to a numeric value and mapped down to an index in the state table in accordance with column descriptors found in the SDN. After the state has been pulled from the table, the value is first checked for end of validation or it is sent to another paragraph for error checking.

.0603-CHECK-RESP-LENGTH
This routine is called to ascertain the length of response for particular questions. The FILE-INDEX variable is used as the indicator. Should this index meet one of the criteria in this routine, the next n characters are moved into the variable used for state table lookup. If all conditions fail, the response is one character only, and is simply moved to the variable.

.0604-CONVERT-DATA
This routine will translate the input data into a column value for table lookup. For a table lookup, please consult the SDN.

.0605-CHECK-SPECIAL-STATES
This paragraph is called to ascertain if the current state requires special evaluation. The three states involved are 40, 80, and 82. This routine was written due to the complexity of the state table to circumvent the necessity of this routine.

.0606-CHECK-ERROR-STATES
This routine will simply check the current state to determine if an error has been detected. The simplistic structure allows easy analysis and maintenance. Should an error state be found, the respective error messages are loaded into the memo record and written to the output file.
This small paragraph receives several calls from 0303 to simply move memo array elements to the output file and write each record. The control variables (array indexes and loop variables) are determined by each calling statement.

This routine will display total counts of client data forms examined for each program on the report. The code will place three programs' results on one line of output before writing to disk. All memo variables are re-initialized after a write.
Appendix G

Hierarchy Plus Input–Process Output (HIPO)
HIPO DOCUMENTATION

System: Inconsistency
Author: Michael L. Richardson
Description: Visual Table of Contents
Name: INCON.CBL

Date: November 27, 1984
Page: 1 of 26

BEGIN-INCON

0200 INITIALIZE-VARIABLES
0201 LOAD-STATE-TABLE
0207 SELECT-ADM-DATES
0208 SELECT-DIS-DATES
0204 SELECT-PPNS
0205 DISPLAY-LOAD-STATE-TABLE
0206 DISPLAY-PPNS
0202 DISPLAY-CMD-LIST
0203 SELECT-AGENCIES

0301 SELECT-PROGRAMS
0302 WRITE-MEMO-HEADER
0303 GET-DATA-BY-PROGRAM

0401 SELECT-DPY-PGM
0402 SELECT-SEL-HELP
0403 GET-CLIENT-DATA
0404 FINISH-CURRENT-MEMO
0405 DISPLAY-PGM-TOTALS

0501 SEARCH-LOAD-551
0502 SEARCH-LOAD-552
0503 SEARCH-LOAD-651
0504 SEARCH-LOAD-652
0505 STATE-LOOKUP

0601 CHECK-FOR-ALL-FORMS
0602 READ-OUT-FORM
0603 CHECK-RESP-LENGTH
0604 CONVERT-CLIENT-DATA
0605 CHECK-SPECIAL-STATES
0606 CHECK-ERROR-STATES

0701 READ-ERROR-DUMP
INPUT
- ALL VARIABLES
- STATE TABLE
- INTERACTIVE INPUT

FUNCTION/PROCESS
- 0200-INITIALIZE-VARIABLES
- 0201-LOAD-STATE-TABLE
- 0202-SELECT-ADM-DATES
- 0203-SELECT-DIS-DATES
- 0204-DISPLAY-STATE-TABLE
- 0205-LOAD-PPNS
- 0206-DISPLAY-DOC
- 0207-DISPLAY-CMD-LIST
- 0208-SELECT-AGENCIES

OUTPUT
- SET VARIABLES (0200)
- VALID STATE TABLE (0201)
- SET DATES (0202, 0203)
- PRINT OUT STATE TABLE (0204)
- LOAD ARRAY OF DISK AREAS (0205)
- PRINT SYSTEM INFORMATION (0206, 0207)
- LOAD PPN-ARRAY (0208)
**System:**  
Author: Michael L. Richardson  
Description: Detail Diagram  

**Name:** 0200-INITIALIZE-VARIABLES  

<table>
<thead>
<tr>
<th><strong>INPUT</strong></th>
<th><strong>FUNCTION/PROCESS</strong></th>
<th><strong>OUTPUT</strong></th>
</tr>
</thead>
</table>
| ALL SYSTEM VARIABLES | IF PIC X, INITIALIZE TO SPACES  
IF PIC 9, INITIALIZE TO ZEROS  
IF "BEGIN" VARIABLE, INITIALIZE TO LOW-VALUES  
IF "END" VARIABLE, INITIALIZE TO HIGH-VALUES | ALL SYSTEM VARIABLES |
Hipo Documentation

System:
Author: Michael L. Richardson
Description: Detail Diagram

Date: November 27, 1984
Page: 4 of 26

Name: 0201-LOAD-STATE-TABLE

**Input**

- Initialized State-Table

**Function/Process**

- Move hard-coded values to each element of the State-Table

**Output**

- State-Table containing correct values
INPUT

ADMISSION OR DISCHARGE DATE ENTERED INTERACTIVELY

FUNCTION/PROCESS

VARIFY CORRECTNESS:
IF MONTH < 1 OR MONTH > 12, ERROR
IF DAY < 1 OR DAY > 31, ERROR
IF BEGINNING DATE > ENDING DATE, ERROR

OUTPUT

BEGIN-ADMISSION-SELECT
BEGIN-DISCHARGE-SELECT
END-ADMISSION-SELECT
END-DISCHARGE-SELECT
INPUT
LOADED STATE-TABLE

FUNCTION/PROCESS
PRINT HEADER
LOOP THRU TABLE AND PRINT VALUES

OUTPUT
PRINTED STATE-TABLE FOR
FOR VERIFICATION

HIPO DOCUMENTATION

System:
Author: Michael L. Richardson
Description: Detail Diagram

Name: 0204-DISPLAY-STATE-TABLE

Date: November 27, 1984
Page: 6 of 26
INPUT
INTERACTIVE INPUT

FUNCTION/PROCESS
VERIFY PROG-NUMBER:
IF PROG-NUMBER = 0,
    GO TO EXIT
IF PROG-NUMBER < 77316 OR
    PROG-NUMBER > 77336,
    ERROR
    ADD PROG-NUMBER TO ARRAY

OUTPUT
PPN-NUMBER-ARRAY
PPN-NUMBER
HIPO DOCUMENTATION

System: 
Author: Michael L. Richardson 
Description: Detail Diagram

Name: 0206-DISPLAY-DOC
       0207-DISPLAY-CMD-LIST

INPUT
HARD CODED TEXT

FUNCTION/PROCESS
DISPLAY LINES AS PRESENTED

OUTPUT
TEXT EXPLAINING SYSTEM COMMAND HELP LIST
HIPO DOCUMENTATION

System:
Author: Michael L. Richardson
Description: Overview Diagram
Name: 0208-SELECT-AGENCIES

Date: November 27, 1984
Page: 9 of 26

INPUT

ADMISSION, DISCHARGE DATES
FLAG FOR PPN's
INTERACTIVE INPUT

FUNCTION/PROCESS

0301-SELECT-PROGRAMS
0302-WRITE-MEMO-HEADER
0303-GET-DATA-BY-PROGRAM

OUTPUT

READY CLIENT DATA FILE FOR READ
BY PROGRAM (0301)
MEMO HEADER (0302)
PRINTOUT CURRENT RUN STATUS
(0303)
CLIENT FILE FOR EVALUATION
(0303)
**Input**
- PROGRAM-FILE
- VARIABLE FOR ALL PROGRAMS
- INTERACTIVE INPUT

**Function/Process**
- 0401-SELECT-ALL-PROGRAMS
- 0402-DPY-PGM-SEL-HELP

**Output**
- LOAD PROGRAM ARRAY WITH PROGRAM ID's (0401)
- PRINT OUT DOCUMENTATION (0402)
**INPUT**

TODAY

MEMO-TABLE

**FUNCTION/PROCESS**

WRITE FIRST 21 MEMO LINES

(HEADER)

**OUTPUT**

OUTPUT-FILE

DISPLAY TIME AND NAME OF CURRENT DISK AREA
HIPO DOCUMENTATION

System:
Author: Michael L. Richardson
Description: Overview Diagram
Name: 0303-GET-DATA-BY-PROGRAM

Date: November 27, 1984
Page: 12 of 26

**INPUT**

CLIENT-DATA-FILE
MEMO ARRAY

**FUNCTION/PROCESS**

0403-GET-CLIENT-DATA
0404-FINISH-CURRENT-MEMO
0405-DISPLAY-PGM-TOTALS

**OUTPUT**

NEXT CLIENT DATA (0403)
COMPLETE CURRENT MEMO (0404, 0405)
INPUT

PROGRAM-FILE
INTERACTIVE RESPONSE
YES-NO

FUNCTION/PROCESS

SEQUENTIALLY READ PROGRAM FILE
AND DISPLAY RECORD DATA.
IF YES-NO = "YES", ADD
PROGRAM DATA TO PROGRAM STACK

OUTPUT

PROGRAM-STACK
System:
Author: Michael L. Richardson
Description: Detail Diagram

Date: November 27, 1984
Page: 14 of 26

Name: 0402-DPY-PGM-SEL-HELP

INPUT
HARD-CODED TEXT

FUNCTION/PROCESS
DISPLAY LINES AS PRESENTED

OUTPUT
COMMAND HELP LIST
HIPO DOCUMENTATION

System: 
Author: Michael L. Richardson 
Description: Overview Diagram 
Name: 0403-GET-CLIENT-DATA

Date: November 27, 1984 
Page: 15 of 26

INPUT
CLIENT DATA RECORD
PROGRAM STACK
ADMISSION, DISCHARGE DATES

FUNCTION/PROCESS
0501-SEARCH-LOAD-551
0502-SEARCH-LOAD-552
0503-SEARCH-LOAD-651
0504-SEARCH-LOAD-652
0505-STATE-LOOKUP

OUTPUT
ARRAY OF LOADED FORM DATA
(0501, 0502, 0503, 0504)
LOGIC STATE BASED ON INPUT
(0505)
FUNCTION/PROCESS

LOAD OUTPUT RECORD AND WRITE
HIPO DOCUMENTATION

System:
Author: Michael L. Richardson
Description: Detail Diagram
Name: 0405-DISPLAY-PGM-TOTALS

Date: November 27, 1984
Page: 17 of 26

INPUT

TOTALS ARRAY
TOTAL-OUTPUT-LINE

FUNCTION/PROCESS

MOVE VALUES IN TOTALS ARRAY TO THE LOCATIONS IN THE TOTAL-OUTPUT-LINE AND WRITE

OUTPUT

OUTPUT-FILE
## HIPO DOCUMENTATION

**System:**
**Author:** Michael L. Richardson
**Description:** Overview Diagram

**Date:** November 27, 1984  
**Page:** 18 of 26

<table>
<thead>
<tr>
<th>INPUT</th>
<th>FUNCTION/PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
</table>
| CLIENT DATA RECORD  
FORM DATA FILES  
FORM-ID-FILE | 0601-CHECK-FOR-ALL-FORMS  
0602-READ-OUT-FORM  
0701-READ-ERROR-DUMP | DETERMINE IF CLIENT HAS FORM  
(0601)  
READ FORM INTO TEMPORARY ARRAY  
(0602)  
STATUS DUMP FOR INVALID READ  
(0701) |
### Overview Diagram

**Name:** 0505-STATE-LOOKUP

**Description:**

**INPUT**
- ARRAY OF CLIENT FORM DATA
- STATE TABLE

**FUNCTION/PROCESS**
- 0603-CHECK-RESP-LENGTH
- 0604-CONVERT-DATA
- 0605-CHECK-SPECIAL-STATES
- 0606-CHECK-ERROR-STATES

**OUTPUT**
- GET A RESPONSE FROM THE INPUT ARRAY AND CONVERT IT TO A TABLE COLUMN INDEX (0603, 0604)
- CHECK LOOKUP STATE FOR A SPECIAL OR ERROR CONDITION (0605, 0606)
HIPO DOCUMENTATION

System:
Author: Michael L. Richardson
Description: Detail Diagram

Date: November 27, 1984
Page: 20 of 26
Name: 0601-CHECK-FOR-ALL-FORMS

INPUT
CLIENT-DATA-RECORD
FORM-NAME-CHECK

FUNCTION/PROCESS
LOOP THROUGH CLIENT DATA RECORD
LOOKING FOR FORM-NAME-CHECK.
IF NOT FOUND, CLIENT-DATA-
INDEX = ∅, ELSE THE INDEX OF
THE FORM POINTER

OUTPUT
CLIENT-DATA-INDEX
<table>
<thead>
<tr>
<th>INPUT</th>
<th>FUNCTION/PROCESS</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPROPRIATE FORM DATA FILE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORM-ID-DATA-NAME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLIENT-DATA-RECORD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVIDE CLIENT-DATA-LOC (CLIENT-DATA-INDEX) BY RECORDS/BLOCK GIVING ACTUAL-KEY REMAINDER-POSITION-KEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF POSITION-KEY ≠ 0, MOVE RECORDS/BLOCK TO POSITION-KEY ELSE ADD 1 TO ACTUAL-KEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPUTE START-PULL = CHAR/RECORD(\times) (POSITION-KEY-1) + 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPUTE END-PULL = CHAR/RECORD(\times) POSITION-KEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>START-PULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>END-PULL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
If FILE-INDEX = an index of a response of more than 1 character, move the values to the appropriate response.
INPUT

RESPONSE

FUNCTION/PROCESS

IF RESPONSE IS ALPHABETIC
CONVERT TO NUMERIC
IF RESPONSE IS NUMERIC AND > 26
MOVE 27 TO COL
IF RESPONSE IS NUMERIC AND = 0
MOVE 29 TO COL
IF RESPONSE IS NUMERIC AND < 26
MOVE RESPONSE TO COL

OUTPUT

COL
INPUT

STATE
552-PRESENT
652-PRESENT
TEST-1-ARRAY

FUNCTION/PROCESS

CHECK FOR SPECIAL STATES 40, 80, AND 82
IF FOUND, MAKE COMPUTATION AND CHECK AS INDICATED BY FSA

OUTPUT

COL
IF STATE IS ERROR STATE, MOVE APPLICABLE ERROR DATA TO MEMO-LINE AND WRITE TO OUTPUT RECORD

OUTPUT-RECORD
HIPO DOCUMENTATION

System: Author: Michael L. Richardson
Description: Detail Diagram
Name: 0701-READ-ERROR-DUMP

Date: November 27, 1984
Page: 26 of 26

**INPUT**

READ-ERROR-DATA

**FUNCTION/PROCESS**

DISPLAY ALL INFORMATION IN READ-ERROR-DATA

**OUTPUT**

HARD COPY OF ERROR LISTING
Appendix H

State Table
Appendix I

INCON Software
IDENTIFICATION DIVISION.
PROGRAM-ID.  INCON

ENVIRONMENT DIVISION.
*COMPANY:  AREAHIDE DATA EVALUATION SERVICES INC.
*AUTHOR:  MICHAEL L. RICHARDSON, EXECUTIVE DIRECTOR
*MACHINE:  DEC SYSTEM-10/88
*LOCATION:  WESTERN MICHIGAN UNIVERSITY, KALAMAZOO, MICHIGAN
*OP SYS:  TOPS-10 VERSION 7.01
*CPU:  KL100 (2)
*LANGUAGE:  COBOL-68
*ACCESS:  REMOTE ON-LINE
*COMPILE DATE:  AUGUST 15, 1984
*IMPL DATE:  SEPTEMBER 1, 1984

THIS SYSTEM IS DESIGNED TO INTERFACE WITH THE ADES DATA COLLECTION SYSTEM, ISAM FILES CONTAINING
CLIENT INFORMATION ARE RELATED TO OTHER ISAM FILES CONTAINING CLIENT SPECIFIC DATA, INCLUDING POINTERS
FOR DIRECT ACCESS TO THE INFORMATION FILES. THE LATTER FILES CONTAIN KEYS WHICH ALLOW FOR DIRECT
LOOKUP FOR EACH CLIENT. FOR FURTHER FILE RELATIONSHIPS, PLEASE CONSULT THE ADES SYSTEM DOCUMENTATION.

BECAUSE OF THESE FILE RELATIONSHIPS, MINIMAL INPUT IS NECESSARY, AS A CLIENT IS EITHER REJECTED OR PASSED
THROUGH FOR ANALYSIS BASED ON ADMISSION AND DISCHARGE DATES FOUND AFTER THE FIRST DIRECT READ. SHOULD A
CLIENT FALL WITHIN THE PARAMETERS ESTABLISHED, INPUT FROM VARIOUS FILES WILL BE DONE IF AND ONLY IF THE
CLIENT SPECIFIC DATA FILE CONTAINS A POINTER TO THE PARTICULAR INFORMATION FILE. THEN EACH CLIENT SPECIFIC
DATA RECORD IS READ ONCE, AND THE POTENTIAL EXISTS FOR UP TO FOUR ADDITIONAL READS PER CLIENT.
THE INCONSISTENCY PROJECT WILL EXAMINE ALL CLIENT INFORMATION AVAILABLE. SHOULD A CLIENT NOT
HAVE ONE OR MORE OF THE FOUR FORMS REQUIRED FOR NORMAL PROCESSING, THE INCONSISTENCY EVALUATION PER-
FORMED BY THE SYSTEM WILL NOT RECORD ANY ERRORS, BUT RATHER 'DROP THROUGH' THAT PARTICULAR DATA SET.
THIS ENSURES ALL ERRORS RECORDED ARE DUE TO INCORRECT DATA COLLECTION OR ENTRY AND NOT THE RESULT OF
MISSING CLIENT DATA.

METHOD OF VERIFYING CONSISTENT DATA WILL BE TABLE-LOOKUP. THIS PROCESS WILL CLOSELY PARALLEL THE
SYNTACTICAL ANALYSIS PERFORMED IN COMPILERS, USING LEXICAL ANALYSIS TECHNIQUES. THIS PROCESS SHOULD
MINIMIZE THE TIME REQUIRED TO EXAMINE DATA ONCE IT HAS BEEN INPUT TO THE SOFTWARE.
ALL CLIENT INFORMATION WILL BE READ INTO THE SYSTEM AND THE ELEMENTS TO BE ANALYZED WILL BE MOVED INTO A
LIST FOR USE IN TABLE TRAVERSAL. THIS PROCESS OF CREATING A LIST OF ALL ELEMENTS TO BE EXAMINED
CIRCUMVENTS THE NEED FOR A COMPLICATED AND CUMBERSOME FILTERING ALGORITHM.
THE STATE TABLE TO BE USED WILL BE A DETERMINISTIC FINITE AUTOMATA. PROVEN TECHNIQUES WILL BE USED TO
MINIMIZE THE TABLE ONCE INITIAL DESIGN IS COMPLETE. ALL Incoming ELEMENTS WILL BE TRANSLATED TO AN INDEX
BASED ON DEFINITIONS DEvised. THIS INFORMATION WILL BECOME PART OF THE SYSTEM DOCUMENTATION NOTEBOOK (SDN).
The DOCUMENT WILL CONTAIN ALL INFORMATION NECESSARY TO UNDERSTAND, TROUBLESHOOT, AND UPDATE THE SYSTEM.
The OUTPUT SHALL BE WRITTEN TO DISK TAKING FULL ADVANTAGE OF THE 'SCRATCHPAD' FACILITY CURRENTLY AVAILABLE
ON THE EXISTING COMPUTER SYSTEM. THE MEMOS IN THIS FILE ARE PRECEDED BY A PAGE-POP FLAG FOR LINE PRINTER
OUTPUT. THE LOCAL PROGRAM TPRINT WILL BE USED TO PRODUCE ON-LINE MEMOS FOR DISTRIBUTION TO THE
COORDINATING AGENCIES. THE LINE PRINTER COPY WILL BE MAINTAINED AT THE ADES OFFICE FOR HISTORICAL
PURPOSES.

CONSTRAINTS

THE PROJECT, DUE TO THE NEED TO INTERFACE WITH THE ADES DATA BASE, SEVERAL INTERNALLY DEVELOPED ROUTINES,
AND THE COMPUTER SYSTEM AT WESTERN MICHIGAN UNIVERSITY (WMU), MUST BE DEVELOPED ON A DEC SYSTEM-10/88 USING
TOPS-10 IN COBOL-68. ALL ADES DATA IS STORED IN USER DEFINED ISAM FILES. THIS SYSTEM HAS BEEN DESIGNED
TO INTERFACE WITH THIS DATA BASE SYSTEM. SEVERAL MACRO ROUTINES FROM THE WMU BUSINESS COLLEGE ARE ALSO
USED TO EXPEDITE DATA ACCESS, PARTICULARLY ACROSS USER BOUNDARIES. THIS PROJECT WILL FALL UNDER THE ADES
ENVIRONMENT, AND THE CONTROL ROUTINES MUST BE ACCESSSED FOR RECORD KEEPING AT VARIOUS RUN STAGES. THESE
ROUTINES, ALONG WITH THE ENTIRE ADES SYSTEM, HAVE BEEN DESIGNED IN COBOL-68. FOR THIS REASON AND THE
FACT THAT COBOL-68 AND COBOL-74 ARE INCOMPATIBLE, THIS PROJECT SHALL BE WRITTEN IN COBOL-68.

* THE INTERFORM CONSISTENCY CHECKS MADE BY THIS SYSTEM ARE DESIGNED FOR THE CURRENT SET OF ADMISSION AND
* DISCHARGE FORMS ONLY. ALL HISTORIC DATA STORED IN OLDER FORMATS WILL NOT BE ANALYZED IN THIS SYSTEM.
* SHOULD THIS PROCESS BE NECESSARY, THE ROUTINES DELINeated IN THE ADES SYSTEM DOCUMENTATION MANUAL WILL
* NEED TO BE ENACTED TO TEMPORARILY TRANSLATE ALL DATA TO THE CURRENT FORMAT. THE FOLLOWUP FORMS USED IN THE
* ADES SYSTEM (SET OF THREE) ARE NOT ANALYZED DUE TO THE METHODS ATTEMPTED IN DATA GATHERING AND THE FACT
* THAT THE MICHIGAN OFFICE OF SUBSTANCE ABUSE SERVICES CURRENTLY DEFINES FOLLOWUP TO BE A PROJECT (I.E.
* SUBJECT TO CHANGE).
* DEVELOPMENT OF THIS SYSTEM SHALL BE COMPLETED IN ACCORDANCE WITH THE PROGRAMMING STANDARDS DEVELOPED BY
* ADES.

** LEVEL 1 ALGORITHM **

BEGIN
  DETERMINE SEARCH CONDITIONS (DATES, DISK AREAS DESIRED)
  LOOP THRU DISK AREAS (CA) UNTIL OUT
    DISPLAY AND WRITE MEMO-HEADER
  LOOP THRU LIST OF SUBSTANCE ABUSE PROGRAMS
    SEQUENTIALLY READ CLIENT DATA FILES
    LOOP THRU FORM-DATA-FILES
      IF FORM IS AVAILABLE READ INTO ARRAY
    END FORM-DATA-FILE LOOP
  TEST DATA USING STATE TABLE
    DISPLAY AND WRITE ERRORS FOUND
  END PROGRAM LOOP
  DISPLAY AND WRITE MEMO-FOOTING
END
  END CA LOOP

** AS ENHANCEMENTS ARE MADE TO THE ADES SYSTEM, THIS SOFTWARE MUST BE MODIFIED TO NOT ONLY REMAIN
* COMPATIBLE WITH THE SYSTEM, BUT MUST BE CHANGED TO MAINTAIN ITS CORRECTNESS. THIS IS ACCOMPLISHED
* THROUGH CHANGES TO THE STATE TABLE. BY OBTAINING A 'LIST' OF THE CURRENT TABLE AND VERIFYING
* CHANGES ON THE STATE TABLE LOGIC CHARTS, THE ROUTINE 0201-LOAD-STATE-TABLE CAN BE EASILY MODIFIED
* TO MEET THESE CHANGES.
* WHENEVER ANY CHANGE TO THIS SOFTWARE IS NECESSARY, THE CONTROL DECK INCON.TST SHALL BE EXECUTED. THIS
* DECK WILL INVOKE THIS SOFTWARE AND BLACKBOX TEST THE SYSTEM FOR CORRECTNESS. INCON.TST WAS DEVELOPED
* SPECIFICALLY FOR THIS PURPOSE.
* WHEN UPDATING THIS SOFTWARE, THE SDN MUST BE CONSULTED AND ANY CHANGES MADE DOCUMENTED IN THE SDN
* USING THE PROTOCOL FOUND IN THE ADES PROGRAMMING STANDARDS.

********************************************************************************************
CONFIGURATION SECTION.
SPECIAL-VALUES.
  CHANNEL (1) IS TOP-OF-PAGE.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT PROGRAM-FILE ASSIGN TO DSK
    RECORDING MODE IS SIXBIT
    ACCESS MODE IS INDEXED
    RECORD KEY IS PROGRAM-RECORD-KEY
    SYMBOLIC KEY IS PROGRAM-SEARCH-KEY.
  SELECT CLIENT-DATA-FILE ASSIGN TO DSK
    RECORDING MODE IS SIXBIT
ACCESS MODE IS INDEXED
RECORD KEY IS CLIENT-DATA-RECORD-KEY
SYMBOLIC KEY IS CLIENT-DATA-SEARCH-KEY.

SELECT FORM-ID-FILE ASSIGN TO DSK
RECORDING MODE IS SIXBIT
ACCESS MODE IS INDEXED
RECORD KEY IS FORM-ID-RECORD-KEY
SYMBOLIC KEY IS FORM-ID-SEARCH-KEY.

SELECT FORM-DATA-FILE ASSIGN TO DSK
RECORDING MODE IS SIXBIT
ACCESS MODE IS RANDOM
FILE LIMITS ARE 1 THRU 99990
ACTUAL KEY IS FORM-DATA-FILE-ACTUAL-KEY.

SELECT OUTPUT-FILE ASSIGN TO DSKC
RECORDING MODE IS ASCII
ACCESS MODE IS SEQUENTIAL.

**********************************************************************
DATA DIVISION.
FILE SECTION.
* BLOCKING FACTORS
*  TOTAL CHAR = 127
*  DATA FILE = 11
*  INDEX FILE = 42N42
FD PROGRAM-FILE VALUE OF ID IS 'PROGRMDX'
USER-NUMBER IS WHOLE-NUMBER
BLOCK CONTAINS 4 RECORDS.
  01 PROGRAM-REC.
     02 PROGRAM-RECORD-KEY PIC X(2).
     02 PROGRAM-NAME PIC X(4S).
     02 PROGRAM-LICENSE-NUM PIC X(6).
     02 PROGRAM-REPORTING-TYPE.
     03 PROGRAM-DRUG PIC 9V9999.
     03 PROGRAM-ALCOHOL PIC 9V9999.
     02 PROGRAM-BASE-DATE.
     03 PROGRAM-BASE-DATE-YEAR PIC 9(2).
     03 PROGRAM-BASE-DATE-MONTH PIC 9(2).
     02 PROGRAM-SHELL.
     03 PROGRAM-SHELL-ID PIC XX OCCURS 2 TIMES.
     02 PROGRAM-ALLOCATE.
     03 PROGRAM-ALLOCATE-ID PIC XX OCCURS 5 TIMES.
     02 PROGRAM-RATIO-TABLE.
     03 PROGRAM-RATIO-ITEM PIC X OCCURS 96 TIMES.
* BLOCKING FACTORS
*  CHAR=127
*  DATA=11
*  KEY=UX.18
*  INDEX=31
FD CLIENT-DATA-FILE VALUE OF IDENTIFICATION 'CLIDATIDX'
USER-NUMBER IS WHOLE-NUMBER
BLOCK CONTAINS 11 RECORDS.
  01 CLIENT-DATA-RECORD.
     02 CLIENT-DATA-RECORD-KEY.
03  CLIENT-DATA-PROGRAM PIC X(2).
03  CLIENT-DATA-SERVICE PIC 99.
03  CLIENT-DATA-ID PIC X(8).
03  CLIENT-DATA-ADMISSION.
   04  CLIENT-DATA-ADMISSION-YEAR PIC XX.
   04  CLIENT-DATA-ADMISSION-MONTH PIC XX.
   04  CLIENT-DATA-ADMISSION-DAY PIC XX.
02  CLIENT-DATA-RECORD-DATE.
   03  CLIENT-DATA-DESC PIC XX.
   03  CLIENT-DATA-STATUS PIC X.
   03  CLIENT-DATA-ASUB.
      04  CLIENT-DATA-ASUB-YEAR PIC XX.
      04  CLIENT-DATA-ASUB-MONTH PIC XX.
      04  CLIENT-DATA-ASUB-DAY PIC XX.
   03  CLIENT-DATA-DISCHARGE.
      04  CLIENT-DATA-DISCHARGE-YEAR PIC XX.
      04  CLIENT-DATA-DISCHARGE-MONTH PIC XX.
      04  CLIENT-DATA-DISCHARGE-DAY PIC XX.
   03  CLIENT-DATA-DSTATUS PIC X.
   03  CLIENT-DATA-DSUB.
      04  CLIENT-DATA-DSUB-YEAR PIC XX.
      04  CLIENT-DATA-DSUB-MONTH PIC XX.
      04  CLIENT-DATA-DSUB-DAY PIC XX.
   03  CLIENT-DATA-FOLLOWUP.
      04  CLIENT-DATA-FOLLOWUP-YEAR PIC XX.
      04  CLIENT-DATA-FOLLOWUP-MONTH PIC XX.
      04  CLIENT-DATA-FOLLOWUP-DAY PIC XX.
   03  CLIENT-DATA-FSTATUS PIC X.
   03  CLIENT-DATA-FSUB.
      04  CLIENT-DATA-FSUB-YEAR PIC XX.
      04  CLIENT-DATA-FSUB-MONTH PIC XX.
      04  CLIENT-DATA-FSUB-DAY PIC XX.
   03  CLIENT-DATA-POINTER-COUNTER PIC 9(2).
   03  CLIENT-DATA-POINTER.
      04  CLIENT-DATA-ARRAY OCCURS 9 TIMES.
   05  CLIENT-DATA-FORM PIC X(3).
   05  CLIENT-DATA-LOC PIC 9(5).

* BLOCKING FACTORS
  * CHAR = 23
  * DATA = 25
  * KEY = UX 3
  * INDEX FILE = 42
FD FORM-ID-FILE VALUE OF ID IS ‘FORMIDIX’
BLOCK CONTAINS 25 RECORDS.
01  FORM-ID-RECORD.
   02  FORM-ID-RECORD-KEY.
   03  FORM-ID-RECORD-ITEM PIC X(3).
   02  FORM-ID-RECORD-DATA.
      03  FORM-ID-RECORDS-BLOCK PIC 9(2).
      03  FORM-ID-CHAR-RECORD PIC 9(3).
      03  FORM-ID-NAME PIC X(6).
      03  FORM-ID-QUESTION-START PIC 9(2).
03 FORM-ID-QUESTION-END PIC 9(2).
03 FORM-ID-FORM-TYPE PIC X.
03 FORM-ID-DEPENDENT-FORM PIC X(3).
03 FORM-ID-ACCESS-STATUS PIC X(1).

* BLOCKING FACTORS
* CHAR=762
* DATA = 1
FD FORM-DATA-FILE VALUE OF IDENTIFICATION IS FORM-DATA-NAME-FILE
USER-NUMBER IS WHOLE-NUMBER
BLOCK CONTAINS 1 RECORDS.
01 FORM-DATA-RECORD.
  02 FORM-DATA-RECORD-ITEM PIC X(1) OCCURS 762.
FD OUTPUT-FILE VALUE OF ID IS 'INCOM FIL'.
  01 OUTPUT-RECORD PIC X(120).

****************************************************************************************
WORKING-STORAGE SECTION.
**THE FOLLOWING VARIABLES ARE USED AS FILE SEARCH KEYS NEEDED IN COBOL-68**
  01 PROGRAM-SEARCH-KEY.
    02 PROGRAM-SEARCH-ID PIC X(2).
  01 CLIENT-DATA-SEARCH-KEY.
    02 CLIENT-DATA-SEARCH-PROGRAM PIC X(2).
    02 CLIENT-DATA-SEARCH-SERVICE PIC X(2).
    02 CLIENT-DATA-SEARCH-ID PIC X(109).
  01 CLIENT-DATA-SEARCH-ADMISSION.
    04 CLIENT-DATA-SEARCH-YEAR PIC X(2).
    04 CLIENT-DATA-SEARCH-MONTH PIC X(2).
    04 CLIENT-DATA-SEARCH-DAY PIC X(2).
  01 FORM-DATA-SEARCH-KEY.
    02 FORM-DATA-SEARCH-ITEM PIC X(3).
  01 SAVE-CLIENT-KEY.
    02 SAVE-PROGRAM PIC X(2).
    02 SAVE-SERVICE PIC X(2).
    02 SAVE-ID PIC X(109).
  01 FORM-DATA-NAME-FILE.
    02 FORM-DATA-NAME-FILE-NAME PIC X(6).
  02 FORM-DATA-NAME-FILE-EXTENTION PIC X(3) VALUE 'FIL'.
  77 FORM-DATA-FILE-ACCOUNT-KEY PIC 9(5) USAGE COMP.
  77 FORM-DATA-FILE-POSITION-KEY PIC 9(2) USAGE COMP.
  77 START-PULL PIC 9(3) USAGE COMP.
  77 END-PULL PIC 9(3) USAGE COMP.

***VARIABLES USED TO ACCEPT USER RESPONSES***
  01 INCOM-COMMAND PIC XXX.
  77 YES-NO PIC X(3).
  77 GET-DATE PIC X(10).

***VARIABLES USED TO TRAVERSE THE STATE TABLE***
  77 STATE PIC 99.
  77 LAST-STATE PIC 99.
  77 ERROR-STATE PIC 9.
  77 STATE-DISPLAY PIC 99.
  77 EXIT-STATE PIC 99 VALUE 86.

***VARIABLES USED TO TRANSLATE FROM PIC X TO PIC 9 FOR COMPUTATION***
01  CHAR-FIELD PIC XXXXXX JUSTIFIED RIGHT.
01  NUM-FIELD REDEFINES CHAR-FIELD PICTURE IS 999999 USAGE COMP.
77  PIC TEST-VAR PIC 9(3).

***VARIOUS INDICES USED THROUGHOUT THE PROGRAM***
77  FILE-INDEX PIC 99.
77  COLUMN-INDEX PIC 9(2).
77  ROW-INDEX PIC 92.
77  LINE-INDEX PIC 99 USAGE COMP.
77  LINE-ROW PIC 9(7) USAGE COMP.
77  COL PIC 999.

***COUNTERS USED FOR VARIOUS UPPER BOUNDS***
01  ERROR-COUNT PIC 9(10).
01  PROGRAMS-PER-LINE PIC 9 VALUE 3.
77  CLIENT-DATA-INDEX PIC 9(2) USAGE COMP.
77  MEMO-COUNTER PIC 99 USAGE COMP.
77  DISPLAY-COUNTER PIC 999 USAGE COMP.

***FLAGS USED TO REPRESENT VARIOUS CONDITIONS***
77  551-PRESENT PIC X.
77  552-PRESENT PIC X.
77  651-PRESENT PIC X.
77  652-PRESENT PIC X.
77  FORM-NAME-CHECK PIC 9(3).
77  CLIENT-MSG PIC 9 USAGE COMP.

***VARIABLES USED TO RELLOCATE RESPONSES IN THE FORM RESPONSES***
01  RESPONSE-1 PIC X.
01  RESPONSE-2 PIC X.
01  RESPONSE-3 PIC X.
01  RESPONSE-BREAKOUT REDEFINES RESPONSE.
02  RESPONSE-1 PIC X.
02  RESPONSE-2 PIC X.
02  RESPONSE-3 PIC X.

***TWO DIMENSIONAL ARRAY TO STORE THE STATE TABLE***
01  TABLE-FILE-ARRAY.
 02  TABLE-FILE-RECORD-DATA.
 03  TABLE-FILE-ARRAY-STATE OCCURS 86 TIMES.
 04  TABLE-FILE-ARRAY-COL OCCURS 29 TIMES.
 05  TABLE-FILE-ARRAY-COL-ITEM PIC 99.

***ARRAY USED TO STORE CLIENT DATA FOR TABLE TRAVERSAL***
01  TEST-1-RECORD.
 05  TEST-1-ITEM PIC X OCCURS 43 TIMES.

***THE FOLLOWING VARIABLES ARE USED TO DETERMINE _INDEX PPN NUMBERS***
01  ALL-PPN-NUMBER-TABLES.
 02  ALL-PPN-NUMBER-TABLE.
 03  FILLER PIC 9(5) VALUE 77324.
 03  FILLER PIC 9(5) VALUE 77330.
 03  FILLER PIC 9(5) VALUE 77336.
 03  FILLER PIC 9(5) VALUE 77316.
 03  FILLER PIC 9(5) VALUE 77321.
 03  FILLER PIC 9(5) VALUE 77331.
 03  FILLER PIC 9(5) VALUE 77322.
 03  FILLER PIC 9(5) VALUE 77327.
 03  FILLER PIC 9(5) VALUE 77320.
03 FILLER PIC 9(5) VALUE 77332.
02 ALL-PPN-NUM-ARRAY REDEFINES ALL-PPN-NUMBER-TABLE.
03 ALL-PPN-NUMBER-ITEM PIC 9(5) OCCURS 10 TIMES.
01 PPN-NUMBER-TABLES.
02 PPN-NUMBER-ITEM PIC X(5) OCCURS 10 TIMES.
01 PPN-SELECT PIC XXX.
01 PPN-COUNTER PIC 99 USAGE COMP.
01 PROG-DISPLAY PIC X(5).
01 PPN-NUMBER PIC 99 USAGE COMP VALUE 10.
01 PROJ-NUMBER PIC 9(5) USAGE COMP VALUE 77315.
01 PROG-NUMBER PIC 9(5) USAGE COMP.
01 WHOLE-NUMBER PIC 9(10) COMP.

***THE BELOW VARIABLES ARE USED TO HANDLE _STORE PROGRAM INFORMATION***
01 PROGRAM-STACK-TABLE.
02 PROGRAM-STACK-ITEM OCCURS 40 TIMES.
05 PROGRAM-STACK-ID PIC X(2).
05 PROGRAM-STACK-NAME PIC X(45).
01 PROGRAM-SELECT-OPTION PIC XX.
01 ALL-PPGMS PIC XX.
01 PROGRAM-ERRORS PIC 999 USAGE COMP.
01 PROGRAM-COUNTER PIC 99 USAGE COMP.
01 PROGRAM-STACK-COUNTER PIC 99 USAGE COMP.

***THE BELOW ARRAYS CONTAIN THE MEMO DATA FOR DISPLAY***
01 MEMO-TABLE.
02 MEMO-TABLE-VALUES.
05 MEMO-HEADER-1.
10 FILLER PIC X(30) VALUE SPACES.
10 FILLER PIC X(14) VALUE 'A.D.E.S., INC.'.
10 FILLER PIC X(41) VALUE SPACES.
05 MEMO-HEADER-2.
10 FILLER PIC X(25) VALUE SPACES.
10 FILLER PIC X(25) VALUE '831 NORTH WASHINGTON AVE.'.
10 FILLER PIC X(35) VALUE SPACES.
05 MEMO-HEADER-3.
10 FILLER PIC X(25) VALUE SPACES.
10 FILLER PIC X(25) VALUE 'LANSING MICHIGAN 48906'.
10 FILLER PIC X(35) VALUE SPACES.
05 FILLER PIC X(85) VALUE SPACES.
05 FILLER PIC X(85) VALUE SPACES.
05 MEMO-HEADER-4.
10 FILLER PIC X(70) VALUE 'MEMORANDUM'.
10 MS-1 PIC XXX.
10 FILLER PIC X VALUE '—'.
10 MS-2 PIC 99.
05 FILLER PIC X VALUE '—'.
10 MS-3 PIC 99.
10 FILLER PIC X(8) VALUE SPACES.
05 FILLER PIC X(85) VALUE '—'.
05 FILLER PIC X(85) VALUE SPACES.
05 FILLER PIC X(85) VALUE 'FROM: MICHAEL L. RICHARDSON, EXECUTIVE DIRECTOR'.
05 MEMO-HEADER-6.
10 FILLER PIC X(7) VALUE 'TO: '.

05 FILLER PIC X(85) VALUE SPACES.
05 FILLER PIC X(85) VALUE '3. PLEASE HAVE ALL PROGRAMS MAKE CORRECTIONS FOR THESE'.
05 FILLER PIC X(85) VALUE 'INCONSISTENCIES. THANK YOU FOR YOUR ATTENTION.'.
05 FILLER PIC X(85) VALUE SPACES.
05 FILLER PIC X(85) VALUE '3. CONGRATULATIONS! YOUR AGENCY HAS CORRECTLY ENTERED'.
05 FILLER PIC X(85) VALUE 'ALL CLIENT DATA. KEEP UP THE GOOD WORK.'.
02 MEMO-TABLE-REDEFINE REDEFINES MEMO-TABLE-VALUES.
05 MEMO-TABLE-ITEM PIC X(85) OCCURS 35 TIMES.

02 MEMO-LINE.
05 FILLER PIC X(4) VALUE SPACES.
05 MEMO-PID PIC XX.
05 FILLER PIC X(3) VALUE SPACES.
05 MEMO-SC PIC XX.
05 FILLER PIC X(3) VALUE SPACES.
05 MEMO-CID PIC X(8).
05 FILLER PIC X(4) VALUE SPACES.
05 MEMO-FORM PIC X(7).
05 FILLER PIC X(4) VALUE SPACES.
02 FORM-INFO.
05 MEMO-QTN1 PIC X(6).
05 MEMO-FILLER PIC X(19) VALUE 'INCONSISTENT WITH'.
05 MEMO-QTN2 PIC X(6).
05 FILLER PIC X(3) VALUE SPACES.
05 MEMO-CLASS PIC X(31).
02 TOTALS-ARRAY.
02 TOTALS-ARRAY-ITEM OCCURS 40 TIMES.
05 TOTALS-ARRAY-ID PIC X(2).
05 TOTALS-ARRAY-COUNT PIC 9(6).
01 TOTAL-OUTPUT-LINE.
02 TOTAL-OUTPUT-LINE-1.
05 TOTAL-OUTPUT-LINE-1-1.
10 FILLER PIC X(4) VALUE SPACE.
10 FILLER PIC X(5) VALUE 'PID: '.
10 FILLER PIC X(2) VALUE '='.
10 CLIENTS-PROCESSED-1 PIC ZZZZZ9.
05 TOTAL-OUTPUT-LINE-1-2.
10 FILLER PIC X(14) VALUE SPACE.
10 FILLER PIC X(5) VALUE 'PID: '.
10 FILLER PIC X(2) VALUE '='.
10 CLIENTS-PROCESSED-2 PIC ZZZZZ9.
02 TOTAL-OUTPUT-LINE-2.
05 TOTAL-OUTPUT-LINE-2-1.
10 FILLER PIC X(14) VALUE SPACE.
10 FILLER PIC X(5) VALUE 'PID: '.
10 PROCESS-ID-3 PIC X(2).
10 FILLER PIC X(2) VALUE '='.
10 CLIENTS-PROCESSED-3 PIC ZZZZZ9.

***THE BELOW VARIABLES ARE USED TO DETERMINE VARIOUS DATES***

01 COMPUTE-MONTH-TABLE.
   02 FILLER PIC X(3) VALUE 'JAN'.
   02 FILLER PIC X(3) VALUE 'FEB'.
   02 FILLER PIC X(3) VALUE 'MAR'.
   02 FILLER PIC X(3) VALUE 'APR'.
   02 FILLER PIC X(3) VALUE 'MAY'.
   02 FILLER PIC X(3) VALUE 'JUN'.
   02 FILLER PIC X(3) VALUE 'JUL'.
   02 FILLER PIC X(3) VALUE 'AUG'.
   02 FILLER PIC X(3) VALUE 'SEP'.
   02 FILLER PIC X(3) VALUE 'OCT'.
   02 FILLER PIC X(3) VALUE 'NOV'.
   02 FILLER PIC X(3) VALUE 'DEC'.

01 JULIAN-MONTH-TABLE REDEFINES COMPUTE-MONTH-TABLE.
02 JULIAN-MONTH PIC X(3) OCCURS 12 TIMES.

01 ADMISSION-SELECT-VALUES.
   02 BEGIN-ADMISSION-SELECT.
      05 BEGIN-ADM-YEAR PIC 99.
      05 BEGIN-ADM-MONTH PIC 99.
      05 BEGIN-ADM-DAY PIC 99.
   02 END-ADMISSION-SELECT.

01 DISCHARGE-SELECT-VALUES.
   02 BEGIN-DISCHARGE-SELECT.
      05 BEGIN-DIS-YEAR PIC 99.
      05 BEGIN-DIS-MONTH PIC 99.
      05 BEGIN-DIS-DAY PIC 99.
   02 END-DISCHARGE-SELECT.

01 HOLD-DATE.
   02 HOLD-YEAR PIC 9(2).
   02 HOLD-MONTH PIC 9(2).
   02 HOLD-DAY PIC 9(2).

01 TIME-DISPLAY.
   05 TODAYS-DATE.
      10 TODAYS-YEAR PIC 99.
      10 TODAYS-MONTH PIC 99.
      10 TODAYS-DAY PIC 99.
   05 TODAYS-TIME.
      10 HOURS PIC XX.
      10 MINUTES PIC XX.
      10 SECONDS PIC XX.

***THESE VARIABLES ARE USED TO INDICATE READ ERRORS***

01 READ-ERROR-DATA.
   02 READ-ERROR-LCN PIC X(6).
   02 READ-ERROR-KEY PIC X(10).
PROCEDURE DIVISION.

*LEVEL 1

*THIS CONTROL PARAGRAPH CALLS OTHER ROUTINES TO INITIALIZE THE STATE TABLE AND ALL VARIABLES.
*THE MAIN MENU IS ALSO HOUSED IN THIS PARAGRAPH. SHOULD A <BLANK> RETURN BE SENSED, THE SYSTEM IS EXITED FROM HERE. THE FOLLOWING LIST DENOTES THOSE ROUTINES CALLED FROM THIS PARAGRAPH:

* 0200-INITIALIZE-VARIABLES
* 0201-LOAD-STATE-TABLE
* 0202-SELECT-ADM-DATES (CONDITIONAL)
* 0203-SELECT-DIS-DATES (CONDITIONAL)
* 0204-DISPLAY-STATE-TABLE (CONDITIONAL)
* 0205-LOAD-PPNS (CONDITIONAL)
* 0206-DISPLAY-DOC (CONDITIONAL)
* 0207-DISPLAY-CMD-LIST (CONDITIONAL)
* 0208-SELECT-AGENCIES (CONDITIONAL)

0100-BEGIN-INCON.
PERFORM 0200-INITIALIZE-VARIABLES THRU 0200-INITIALIZE-VARIABLES-EXIT.
PERFORM 0201-LOAD-STATE-TABLE THRU 0201-LOAD-STATE-TABLE-EXIT.

MAIN-MENU.
DISPLAY ' '.
DISPLAY 'INCON COMMAND: ', WITH NO ADVANCING.
ACCEPT INCON-COMMAND.
IF INCON-COMMAND Equals SPACES
  GO TO 0100-END-INCON.
IF INCON-COMMAND Equals 'ADM'
  PERFORM 0202-SELECT-ADM-DATES THRU 0202-SELECT-ADM-DATES-EXIT
  DISPLAY ' ',
  GO TO MAIN-MENU.
IF INCON-COMMAND Equals 'DIS'
  PERFORM 0203-SELECT-DIS-DATES THRU 0203-SELECT-DIS-DATES-EXIT
  DISPLAY ' ',
  GO TO MAIN-MENU.
IF INCON-COMMAND Equals 'LIS'
  ENTER MACRO TTYWID USING 132
  PERFORM 0204-DISPLAY-STATE-TABLE THRU 0204-DISPLAY-STATE-TABLE-EXIT
  DISPLAY ' ',
  GO TO MAIN-MENU.
IF INCON-COMMAND Equals 'PPN'
  PERFORM 0205-LOAD-PPNS THRU 0205-LOAD-PPNS-EXIT
  DISPLAY ' ',
  GO TO MAIN-MENU.
IF INCON-COMMAND Equals 'DOC'
  PERFORM 0206-DISPLAY-DOC THRU 0206-DISPLAY-DOC-EXIT
  DISPLAY ' ',
  GO TO MAIN-MENU.

0200-BEGIN-INCON.
PERFORM 0200-INITIALIZE-VARIABLES THRU 0200-INITIALIZE-VARIABLES-EXIT.
PERFORM 0201-LOAD-STATE-TABLE THRU 0201-LOAD-STATE-TABLE-EXIT.

MAIN-MENU.
DISPLAY ' '.
DISPLAY 'INCON COMMAND: ', WITH NO ADVANCING.
ACCEPT INCON-COMMAND.
IF INCON-COMMAND Equals SPACES
  GO TO 0100-END-INCON.
IF INCON-COMMAND Equals 'ADM'
  PERFORM 0202-SELECT-ADM-DATES THRU 0202-SELECT-ADM-DATES-EXIT
  DISPLAY ' ',
  GO TO MAIN-MENU.
IF INCON-COMMAND Equals 'DIS'
  PERFORM 0203-SELECT-DIS-DATES THRU 0203-SELECT-DIS-DATES-EXIT
  DISPLAY ' ',
  GO TO MAIN-MENU.
IF INCON-COMMAND Equals 'LIS'
  ENTER MACRO TTYWID USING 132
  PERFORM 0204-DISPLAY-STATE-TABLE THRU 0204-DISPLAY-STATE-TABLE-EXIT
  DISPLAY ' ',
  GO TO MAIN-MENU.
IF INCON-COMMAND Equals 'PPN'
  PERFORM 0205-LOAD-PPNS THRU 0205-LOAD-PPNS-EXIT
  DISPLAY ' ',
  GO TO MAIN-MENU.
IF INCON-COMMAND Equals 'DOC'
  PERFORM 0206-DISPLAY-DOC THRU 0206-DISPLAY-DOC-EXIT
  DISPLAY ' ',
  GO TO MAIN-MENU.
IF INCON-COMMAND EQUALS 'HEL'
    PERFORM 0207-DISPLAY-CMD-LIST THRU
            0207-DISPLAY-CMD-LIST-EXIT
    DISPLAY ',',
    GO TO MAIN-MENU.
IF (BEGIN-ADMISSION-SELECT EQUALS LOW-VALUES)
  OR (BEGIN-DISCHARGE-SELECT EQUALS LOW-VALUES)
    DISPLAY 'ERROR:'
    DISPLAY 'REASON: ALL BEGINNING DATES NOT SET'
    GO TO MAIN-MENU.
IF INCON-COMMAND EQUALS 'INC'
    PERFORM 0208-SELECT-AGENCIES THRU 0208-SELECT-AGENCIES-EXIT
    DISPLAY ',',
    MOVE TODAY TO TIME-DISPLAY
    DISPLAY 'TIME: ',HOURS,' ',MINUTES,' ',SECONDS
    GO TO MAIN-MENU.
DISPLAY 'ERROR:'
   DISPLAY 'REASON: UNRECOGNIZED INCON-COMMAND.'
   GO TO MAIN-MENU.
0100-END-INCON.
STOP RUN.
*****************************************************************************
LEVEL 2
*****************************************************************************
* THIS ROUTINE, CALLED FROM 0100, INITIALIZES ALL VARIABLES FOR THE CURRENT RUN.
0200-INITIALIZE-VARIABLES.
   MOVE SPACES TO INCON-COMMAND.
   YES-NO,
   GET-DATE,
   CHAR-FIELD,
   551-PRESENT,
   552-PRESENT,
   651-PRESENT,
   652-PRESENT,
   RESPONSE,
   RESPONSE-BREAKOUT,
   TEST-1-RECORD,
   PPN-SELECT,
   PPD-DISPLAY,
   PROGRAM-STACK-TABLE,
   PROGRAM-SELECT-OPTION,
   ALL-PPNS,
   TODAY'S-TIME,
   READ-ERROR-DATA.
   MOVE ZEROS TO FORM-FILE-POINTS-KEY,
   FORM-FILE-POSITION-KEY,
   START-PULL,
   END-PULL,
   STATE,
   LAST-STATE,
   ERROR-STATE,
   STATE-DISPLAY,
TEST-VAR,
FILE-INDEX,
COLUMN-INDEX,
ROW-INDEX,
LINE-INDEX,
COL,
ERROR-COUNT,
CLIENT-DATA-INDEX,
MEMO-COUNTER,
DISPLAY-COUNTER,
FORM-NAME-CHECK,
CLIENT-MSG,
RESPONSE-2N,
PPN-COUNTER,
PROC-NUMBER,
WHOLE-NUMBER,
PROGRAM-ERRORS,
PROGRAM-COUNTER,
PROGRAM-STACK-COUNTER.

MOVE HIGH-VALUES TO END-ADMISSION-SELECT,
END-DISCHARGE-SELECT,
TODAYS-DATE.
MOVE LOW-VALUES TO BEGIN-ADMISSION-SELECT,
BEGIN-DISCHARGE-SELECT.
MOVE 1 TO COLUMN-INDEX.
PERFORM INITIALIZE-STATE-TABLE VARYING STATE FROM 1 BY 1
UNTIL STATE > 84.
GO TO 0200-INITIALIZE-VARIABLES-EXIT.
INITIALIZE-STATE-TABLE:
IF COLUMN-INDEX > 29
MOVE 1 TO COLUMN-INDEX
ELSE
MOVE 0 TO TABLE-FILE-ARRAY-COL-ITEM(STATE,COLUMN-INDEX)
ADD 1 TO COLUMN-INDEX
GO TO INITIALIZE-STATE-TABLE.
0200-INITIALIZE-VARIABLES-EXIT.
EXIT.

**********************************************************************
**THIS PARAGRAPH, CALLED FROM 0100, ALTHOUGH EXCEPTIONALLY LONG, IS QUITE STRAIGHTFORWARD. EACH**
**ELEMENT OF THE STATE TABLE IS LOADED AS NECESSARY. WHERE POSSIBLE, ONE LINE INTERNAL PARAGRAPHS ARE**
**LOOVED THRU TO REDUCE THE CODING LINES NECESSARY. THE CODE IS GROUPED BY ROW OF THE STATE TABLE,**
**THUS PROVIDING EASY TROUBLESHOOTING.**
0201-LOAD-STATE-TABLE.
LOAD-STATE-VALUES.
MOVE 2 TO TABLE-FILE-ARRAY-COL-ITEM(1,1),TABLE-FILE-ARRAY-COL-ITEM(1,2),
MOVE 3 TO TABLE-FILE-ARRAY-COL-ITEM(1,3),TABLE-FILE-ARRAY-COL-ITEM(1,28),TABLE-FILE-ARRAY-COL-ITEM(1,4),
PERFORM R2 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 15.
MOVE 6 TO TABLE-FILE-ARRAY-COL-ITEM(2,9),TABLE-FILE-ARRAY-COL-ITEM(2,12),TABLE-FILE-ARRAY-COL-ITEM(2,15),
MOVE 4 TO TABLE-FILE-ARRAY-COL-ITEM(2,28).
PERFORM R3 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 16.
MOVE 6 TO TABLE-FILE-ARRAY-COL-ITEM(3.1),TABLE-FILE-ARRAY-COL-ITEM(3.9),
     TABLE-FILE-ARRAY-COL-ITEM(3.12),TABLE-FILE-ARRAY-COL-ITEM(3.15).
MOVE 5 TO TABLE-FILE-ARRAY-COL-ITEM(3.28).
PERFORM R4 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 28.
MOVE 7 TO TABLE-FILE-ARRAY-COL-ITEM(4.29).
PERFORM R5 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 28.
MOVE 8 TO TABLE-FILE-ARRAY-COL-ITEM(5.27),
     TABLE-FILE-ARRAY-COL-ITEM(5.28),
     TABLE-FILE-ARRAY-COL-ITEM(5.29).
PERFORM R6 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
PERFORM R7 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
MOVE 9 TO TABLE-FILE-ARRAY-COL-ITEM(8.1),TABLE-FILE-ARRAY-COL-ITEM(8.2).
MOVE 10 TO TABLE-FILE-ARRAY-COL-ITEM(8.2).
PERFORM R8 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
PERFORM R9 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
MOVE 11 TO TABLE-FILE-ARRAY-COL-ITEM(10.29).
PERFORM R10 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
MOVE 12 TO TABLE-FILE-ARRAY-COL-ITEM(12.1),TABLE-FILE-ARRAY-COL-ITEM(12.28).
MOVE 13 TO TABLE-FILE-ARRAY-COL-ITEM(12.28).
MOVE 14 TO TABLE-FILE-ARRAY-COL-ITEM(12.2).
MOVE 15 TO TABLE-FILE-ARRAY-COL-ITEM(12.3),TABLE-FILE-ARRAY-COL-ITEM(12.4).
MOVE 16 TO TABLE-FILE-ARRAY-COL-ITEM(12.5).
MOVE 17 TO TABLE-FILE-ARRAY-COL-ITEM(12.6).
MOVE 18 TO TABLE-FILE-ARRAY-COL-ITEM(13.1).
MOVE 19 TO TABLE-FILE-ARRAY-COL-ITEM(13.2).
MOVE 20 TO TABLE-FILE-ARRAY-COL-ITEM(13.3),TABLE-FILE-ARRAY-COL-ITEM(13.4).
MOVE 21 TO TABLE-FILE-ARRAY-COL-ITEM(13.5),TABLE-FILE-ARRAY-COL-ITEM(13.8).
MOVE 22 TO TABLE-FILE-ARRAY-COL-ITEM(13.9).
MOVE 23 TO TABLE-FILE-ARRAY-COL-ITEM(13.10),TABLE-FILE-ARRAY-COL-ITEM(13.7).
MOVE 24 TO TABLE-FILE-ARRAY-COL-ITEM(13.11),TABLE-FILE-ARRAY-COL-ITEM(13.6),TABLE-FILE-ARRAY-COL-ITEM(13.9).
MOVE 25 TO TABLE-FILE-ARRAY-COL-ITEM(13.28).
PERFORM R11 VARYING COLUMN-INDEX FROM 5 BY 1 UNTIL COLUMN-INDEX > 8.
MOVE 14 TO TABLE-FILE-ARRAY-COL-ITEM(14.2),TABLE-FILE-ARRAY-COL-ITEM(14.28).
MOVE 15 TO TABLE-FILE-ARRAY-COL-ITEM(14.4).
MOVE 16 TO TABLE-FILE-ARRAY-COL-ITEM(14.10).
MOVE 17 TO TABLE-FILE-ARRAY-COL-ITEM(14.9).
MOVE 18 TO TABLE-FILE-ARRAY-COL-ITEM(15.2),TABLE-FILE-ARRAY-COL-ITEM(15.28).
MOVE 19 TO TABLE-FILE-ARRAY-COL-ITEM(15.4).
MOVE 20 TO TABLE-FILE-ARRAY-COL-ITEM(15.5),TABLE-FILE-ARRAY-COL-ITEM(15.8).
MOVE 21 TO TABLE-FILE-ARRAY-COL-ITEM(15.10).
MOVE 22 TO TABLE-FILE-ARRAY-COL-ITEM(15.1).
MOVE 23 TO TABLE-FILE-ARRAY-COL-ITEM(15.6),TABLE-FILE-ARRAY-COL-ITEM(15.9).
PERFORM R12 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 9.
MOVE 21 TO TABLE-FILE-ARRAY-COL-ITEM(16.1),TABLE-FILE-ARRAY-COL-ITEM(16.8),
     TABLE-FILE-ARRAY-COL-ITEM(16.28).
MOVE 22 TO TABLE-FILE-ARRAY-COL-ITEM(16.10).
PERFORM R13 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 9.
MOVE 21 TO TABLE-FILE-ARRAY-COL-ITEM(17.1),TABLE-FILE-ARRAY-COL-ITEM(17.28).
MOVE 22 TO TABLE-FILE-ARRAY-COL-ITEM(17.10).
PERFORM R14 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 14.
MOVE 26 TO TABLE-FILE-ARRAY-COL-ITEM(18.2),
     TABLE-FILE-ARRAY-COL-ITEM(18.4),
     TABLE-FILE-ARRAY-COL-ITEM(18.28).
PERFORM R19 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 14.
MOVE 26 TO TABLE-FILE-ARRAY-COL-ITEM(19,1),TABLE-FILE-ARRAY-COL-ITEM(19,2).
PERFORM R20 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 14.
MOVE 26 TO TABLE-FILE-ARRAY-COL-ITEM(20,2),TABLE-FILE-ARRAY-COL-ITEM(20,4).
PERFORM R21 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 14.
MOVE 26 TO TABLE-FILE-ARRAY-COL-ITEM(21,2),TABLE-FILE-ARRAY-COL-ITEM(21,6).
PERFORM R22 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 14.
MOVE 26 TO TABLE-FILE-ARRAY-COL-ITEM(22,7),TABLE-FILE-ARRAY-COL-ITEM(22,8).
PERFORM R23 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
PERFORM R24 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
PERFORM R25 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
GO TO LOAD-551-552-STATES.
R2.
MOVE 4 TO TABLE-FILE-ARRAY-COL-ITEM(2,COLUMN-INDEX).
R3.
MOVE 5 TO TABLE-FILE-ARRAY-COL-ITEM(3,COLUMN-INDEX).
R4.
MOVE 9 TO TABLE-FILE-ARRAY-COL-ITEM(4,COLUMN-INDEX).
R5.
MOVE 7 TO TABLE-FILE-ARRAY-COL-ITEM(5,COLUMN-INDEX).
R6.
MOVE 9 TO TABLE-FILE-ARRAY-COL-ITEM(6,COLUMN-INDEX).
R7.
MOVE 9 TO TABLE-FILE-ARRAY-COL-ITEM(7,COLUMN-INDEX).
R8.
MOVE 10 TO TABLE-FILE-ARRAY-COL-ITEM(9,COLUMN-INDEX).
R9.
MOVE 12 TO TABLE-FILE-ARRAY-COL-ITEM(10,COLUMN-INDEX).
R10.
MOVE 12 TO TABLE-FILE-ARRAY-COL-ITEM(11,COLUMN-INDEX).
R11.
MOVE 21 TO TABLE-FILE-ARRAY-COL-ITEM(12,COLUMN-INDEX).
R12.
MOVE 23 TO TABLE-FILE-ARRAY-COL-ITEM(13,COLUMN-INDEX).
R13.
MOVE 23 TO TABLE-FILE-ARRAY-COL-ITEM(14,COLUMN-INDEX).
R14.
MOVE 24 TO TABLE-FILE-ARRAY-COL-ITEM(15,COLUMN-INDEX).
R15.
MOVE 24 TO TABLE-FILE-ARRAY-COL-ITEM(16,COLUMN-INDEX).
R16.
MOVE 26 TO TABLE-FILE-ARRAY-COL-ITEM(17,COLUMN-INDEX).
R17.
MOVE 26 TO TABLE-FILE-ARRAY-COL-ITEM(18,COLUMN-INDEX).
R18.
MOVE 24 TO TABLE-FILE-ARRAY-COL-ITEM(19,COLUMN-INDEX).
R19.
MOVE 24 TO TABLE-FILE-ARRAY-COL-ITEM(20,COLUMN-INDEX).
R20.
MOVE 24 TO TABLE-FILE-ARRAY-COL-ITEM(21,COLUMN-INDEX).
R21.
MOVE 24 TO TABLE-FILE-ARRAY-COL-ITEM(22,COLUMN-INDEX).
R22.
MOVE 25 TO TABLE-FILE-ARRAY-COL-ITEM(23,COLUMN-INDEX).
R23.
MOVE 25 TO TABLE-FILE-ARRAY-COL-ITEM(24,COLUMN-INDEX).
R24.
MOVE 26 TO TABLE-FILE-ARRAY-COL-ITEM(25,COLUMN-INDEX).
R25.
MOVE 26 TO TABLE-FILE-ARRAY-COL-ITEM(26,COLUMN-INDEX).
LOAD-551-552-STATES.
MOVE 27 TO TABLE-FILE-ARRAY-COL-ITEM(26,1),TABLE-FILE-ARRAY-COL-ITEM(26,2).
MOVE 28 TO TABLE-FILE-ARRAY-COL-ITEM(26,3).
MOVE 29 TO TABLE-FILE-ARRAY-COL-ITEM(26,4).
MOVE 30 TO TABLE-FILE-ARRAY-COL-ITEM(26,5).
MOVE 31 TO TABLE-FILE-ARRAY-COL-ITEM(26,6).
MOVE 32 TO TABLE-FILE-ARRAY-COL-ITEM(26,7).
PERFORM R27 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 27.
MOVE 27 TO TABLE-FILE-ARRAY-COL-ITEM(27,1),TABLE-FILE-ARRAY-COL-ITEM(27,2).
PERFORM R28 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 27.
MOVE 34 TO TABLE-FILE-ARRAY-COL-ITEM(28,1),TABLE-FILE-ARRAY-COL-ITEM(28,2).
PERFORM R29 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 24.
MOVE 33 TO TABLE-FILE-ARRAY-COL-ITEM(29,1),TABLE-FILE-ARRAY-COL-ITEM(29,2).
PERFORM R292 VARYING COLUMN-INDEX FROM 19 BY 1 UNTIL COLUMN-INDEX > 22.
MOVE 34 TO TABLE-FILE-ARRAY-COL-ITEM(29,23),TABLE-FILE-ARRAY-COL-ITEM(29,28).
PERFORM R30 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 22.
MOV 0 TO TABLE-FILE-ARRAY-COL-ITEM(30, 9), TABLE-FILE-ARRAY-COL-ITEM(30, 12).
MOV 33 TO TABLE-FILE-ARRAY-COL-ITEM(30, 23), TABLE-FILE-ARRAY-COL-ITEM(30, 24).
MOV 34 TO TABLE-FILE-ARRAY-COL-ITEM(30, 28).
PERFORM R31 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 24.
MOV 0 TO TABLE-FILE-ARRAY-COL-ITEM(31, 9), TABLE-FILE-ARRAY-COL-ITEM(31, 12).
MOV 34 TO TABLE-FILE-ARRAY-COL-ITEM(31, 20), TABLE-FILE-ARRAY-COL-ITEM(31, 28).
PERFORM R32 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 23.
MOV 0 TO TABLE-FILE-ARRAY-COL-ITEM(32, 9), TABLE-FILE-ARRAY-COL-ITEM(32, 12).
MOV 34 TO TABLE-FILE-ARRAY-COL-ITEM(32, 24), TABLE-FILE-ARRAY-COL-ITEM(32, 28).
PERFORM R33 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 28.
GO TO LOAD-552-STATES.

R27. MOV 33 TO TABLE-FILE-ARRAY-COL-ITEM(27, COLUMN-INDEX).
R28. MOV 33 TO TABLE-FILE-ARRAY-COL-ITEM(26, COLUMN-INDEX).
R29. MOV 33 TO TABLE-FILE-ARRAY-COL-ITEM(25, COLUMN-INDEX).
R30. MOV 34 TO TABLE-FILE-ARRAY-COL-ITEM(30, COLUMN-INDEX).
R31. MOV 33 TO TABLE-FILE-ARRAY-COL-ITEM(31, COLUMN-INDEX).
R32. MOV 33 TO TABLE-FILE-ARRAY-COL-ITEM(32, COLUMN-INDEX).
R33. MOV 34 TO TABLE-FILE-ARRAY-COL-ITEM(33, COLUMN-INDEX).

LOAD-552-STATES.

PERFORM R34 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 24.
MOV 0 TO TABLE-FILE-ARRAY-COL-ITEM(34, 9), TABLE-FILE-ARRAY-COL-ITEM(34, 12).
MOV 35 TO TABLE-FILE-ARRAY-COL-ITEM(34, 20), TABLE-FILE-ARRAY-COL-ITEM(34, 24).
PERFORM R35 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 27.
MOV 38 TO TABLE-FILE-ARRAY-COL-ITEM(35, 28), TABLE-FILE-ARRAY-COL-ITEM(35, 29).
PERFORM R36 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 27.
MOV 37 TO TABLE-FILE-ARRAY-COL-ITEM(36, 28).
PERFORM R37 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
MOV 40 TO TABLE-FILE-ARRAY-COL-ITEM(37, 28).
PERFORM R38 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 27.
MOV 41 TO TABLE-FILE-ARRAY-COL-ITEM(38, 28).
PERFORM R39 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
MOV 41 TO TABLE-FILE-ARRAY-COL-ITEM(40, 25), TABLE-FILE-ARRAY-COL-ITEM(40, 28).
PERFORM R41 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 21.
MOV 0 TO TABLE-FILE-ARRAY-COL-ITEM(41, 9), TABLE-FILE-ARRAY-COL-ITEM(41, 12).
MOV 42 TO TABLE-FILE-ARRAY-COL-ITEM(41, 14).
MOV 43 TO TABLE-FILE-ARRAY-COL-ITEM(41, 23), TABLE-FILE-ARRAY-COL-ITEM(41, 28).
MOV 44 TO TABLE-FILE-ARRAY-COL-ITEM(41, 2), TABLE-FILE-ARRAY-COL-ITEM(41, 3).
PERFORM R42 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 27.
MOV 47 TO TABLE-FILE-ARRAY-COL-ITEM(42, 28).
PERFORM R43 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 26.
PERFORM R44 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 27.
MOV 47 TO TABLE-FILE-ARRAY-COL-ITEM(44, 14), TABLE-FILE-ARRAY-COL-ITEM(44, 25), TABLE-FILE-ARRAY-COL-ITEM(44, 28).
PERFORM R45 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 27.
MOVE 47 TO TABLE-FILE-ARRAY-COL-ITEM(45,14), TABLE-FILE-ARRAY-COL-ITEM(45,28).
PERFORM R46 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
MOVE 0 TO TABLE-FILE-ARRAY-COL-ITEM(47,9), TABLE-FILE-ARRAY-COL-ITEM(47,12).
MOVE 48 TO TABLE-FILE-ARRAY-COL-ITEM(47,20), TABLE-FILE-ARRAY-COL-ITEM(47,24).
MOVE 50 TO TABLE-FILE-ARRAY-COL-ITEM(47,28).
PERFORM R48 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 24.
MOVE 0 TO TABLE-FILE-ARRAY-COL-ITEM(48,8), TABLE-FILE-ARRAY-COL-ITEM(48,12).
MOVE 51 TO TABLE-FILE-ARRAY-COL-ITEM(48,28).
PERFORM R49 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
MOVE 51 TO TABLE-FILE-ARRAY-COL-ITEM(49,20), TABLE-FILE-ARRAY-COL-ITEM(49,24).
MOVE 53 TO TABLE-FILE-ARRAY-COL-ITEM(49,28).
PERFORM R50 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 17.
MOVE 49 TO TABLE-FILE-ARRAY-COL-ITEM(50,20), TABLE-FILE-ARRAY-COL-ITEM(50,24).
MOVE 52 TO TABLE-FILE-ARRAY-COL-ITEM(50,14).
MOVE 55 TO TABLE-FILE-ARRAY-COL-ITEM(50,28).
MOVE 53 TO TABLE-FILE-ARRAY-COL-ITEM(50,25).
MOVE 54 TO TABLE-FILE-ARRAY-COL-ITEM(51,14).
MOVE 55 TO TABLE-FILE-ARRAY-COL-ITEM(51,25).
PERFORM R53 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
MOVE 55 TO TABLE-FILE-ARRAY-COL-ITEM(52,14).
PERFORM R54 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
PERFORM R55 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 23.
MOVE 0 TO TABLE-FILE-ARRAY-COL-ITEM(55,9), TABLE-FILE-ARRAY-COL-ITEM(55,12).
MOVE 56 TO TABLE-FILE-ARRAY-COL-ITEM(55,20), TABLE-FILE-ARRAY-COL-ITEM(55,24).
MOVE 58 TO TABLE-FILE-ARRAY-COL-ITEM(55,28).
MOVE 57 TO TABLE-FILE-ARRAY-COL-ITEM(56,1).
MOVE 59 TO TABLE-FILE-ARRAY-COL-ITEM(56,28).
PERFORM R57 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
MOVE 59 TO TABLE-FILE-ARRAY-COL-ITEM(58,1).
GO TO LOAD-651-STATES.
MOVE 38 TO TABLE-FILE-ARRAY-COL-ITEM(34, COLUMN-INDEX).
R34.
MOVE 36 TO TABLE-FILE-ARRAY-COL-ITEM(34, COLUMN-INDEX).
| R39. | MOVE 41 TO TABLE-FILE-ARRAY-COL-ITEM(39,COLUMN-INDEX). |
| R40. | MOVE 45 TO TABLE-FILE-ARRAY-COL-ITEM(41,COLUMN-INDEX). |
| R41. | MOVE 46 TO TABLE-FILE-ARRAY-COL-ITEM(42,COLUMN-INDEX). |
| R42. | MOVE 47 TO TABLE-FILE-ARRAY-COL-ITEM(43,COLUMN-INDEX). |
| R43. | MOVE 48 TO TABLE-FILE-ARRAY-COL-ITEM(44,COLUMN-INDEX). |
| R44. | MOVE 49 TO TABLE-FILE-ARRAY-COL-ITEM(45,COLUMN-INDEX). |
| R45. | MOVE 50 TO TABLE-FILE-ARRAY-COL-ITEM(46,COLUMN-INDEX). |
| R46. | MOVE 51 TO TABLE-FILE-ARRAY-COL-ITEM(47,COLUMN-INDEX). |
| R47. | MOVE 52 TO TABLE-FILE-ARRAY-COL-ITEM(48,COLUMN-INDEX). |
| R48. | MOVE 53 TO TABLE-FILE-ARRAY-COL-ITEM(49,COLUMN-INDEX). |
| R49. | MOVE 54 TO TABLE-FILE-ARRAY-COL-ITEM(50,COLUMN-INDEX). |
| R50. | MOVE 55 TO TABLE-FILE-ARRAY-COL-ITEM(51,COLUMN-INDEX). |
| R51. | MOVE 56 TO TABLE-FILE-ARRAY-COL-ITEM(52,COLUMN-INDEX). |
| R52. | MOVE 57 TO TABLE-FILE-ARRAY-COL-ITEM(53,COLUMN-INDEX). |
| R53. | MOVE 58 TO TABLE-FILE-ARRAY-COL-ITEM(54,COLUMN-INDEX). |
| R54. | MOVE 59 TO TABLE-FILE-ARRAY-COL-ITEM(55,COLUMN-INDEX). |
| R55. | MOVE 60 TO TABLE-FILE-ARRAY-COL-ITEM(56,COLUMN-INDEX). |
| R56. | MOVE 61 TO TABLE-FILE-ARRAY-COL-ITEM(57,COLUMN-INDEX). |
| R57. | MOVE 62 TO TABLE-FILE-ARRAY-COL-ITEM(58,COLUMN-INDEX). |

LOAD-851-STATES:

MOVE 63 TO TABLE-FILE-ARRAY-COL-ITEM(59,1), TABLE-FILE-ARRAY-COL-ITEM(59,2).
MOVE 64 TO TABLE-FILE-ARRAY-COL-ITEM(59,3), TABLE-FILE-ARRAY-COL-ITEM(59,4).
MOVE 65 TO TABLE-FILE-ARRAY-COL-ITEM(59,5).

PERFORM R60 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 3.
MOVE 66 TO TABLE-FILE-ARRAY-COL-ITEM(60,1).
MOVE 67 TO TABLE-FILE-ARRAY-COL-ITEM(60,2).
MOVE 68 TO TABLE-FILE-ARRAY-COL-ITEM(60,3).

PERFORM R61 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 6.
MOVE 69 TO TABLE-FILE-ARRAY-COL-ITEM(61,1).
MOVE 70 TO TABLE-FILE-ARRAY-COL-ITEM(61,2).
MOVE 71 TO TABLE-FILE-ARRAY-COL-ITEM(61,3).

PERFORM R62 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 9.
MOVE 72 TO TABLE-FILE-ARRAY-COL-ITEM(62,1).
MOVE 73 TO TABLE-FILE-ARRAY-COL-ITEM(62,2).
MOVE 74 TO TABLE-FILE-ARRAY-COL-ITEM(62,3).

PERFORM R63 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 12.
MOVE 75 TO TABLE-FILE-ARRAY-COL-ITEM(63,1), TABLE-FILE-ARRAY-COL-ITEM(63,2).
MOVE 76 TO TABLE-FILE-ARRAY-COL-ITEM(63,3), TABLE-FILE-ARRAY-COL-ITEM(63,4).
MOVE 77 TO TABLE-FILE-ARRAY-COL-ITEM(63,5).

PERFORM R64 VARYING COLUMN-INDEX FROM 2 BY 1 UNTIL COLUMN-INDEX > 16.
MOVE 78 TO TABLE-FILE-ARRAY-COL-ITEM(64,1), TABLE-FILE-ARRAY-COL-ITEM(64,2).
MOVE 79 TO TABLE-FILE-ARRAY-COL-ITEM(64,3), TABLE-FILE-ARRAY-COL-ITEM(64,4).
MOVE 80 TO TABLE-FILE-ARRAY-COL-ITEM(64,5).

PERFORM R65 VARYING COLUMN-INDEX FROM 3 BY 1 UNTIL COLUMN-INDEX > 20.
MOVE 81 TO TABLE-FILE-ARRAY-COL-ITEM(65,1), TABLE-FILE-ARRAY-COL-ITEM(65,2).
MOVE 82 TO TABLE-FILE-ARRAY-COL-ITEM(65,3), TABLE-FILE-ARRAY-COL-ITEM(65,4).
MOVE 83 TO TABLE-FILE-ARRAY-COL-ITEM(65,5).

PERFORM R66 VARYING COLUMN-INDEX FROM 4 BY 1 UNTIL COLUMN-INDEX > 24.
MOVE 84 TO TABLE-FILE-ARRAY-COL-ITEM(66,1), TABLE-FILE-ARRAY-COL-ITEM(66,2).
MOVE 85 TO TABLE-FILE-ARRAY-COL-ITEM(66,3), TABLE-FILE-ARRAY-COL-ITEM(66,4).
MOVE 86 TO TABLE-FILE-ARRAY-COL-ITEM(66,5).

PERFORM R67 VARYING COLUMN-INDEX FROM 5 BY 1 UNTIL COLUMN-INDEX > 29.
MOVE 87 TO TABLE-FILE-ARRAY-COL-ITEM(67,1), TABLE-FILE-ARRAY-COL-ITEM(67,2).
MOVE 88 TO TABLE-FILE-ARRAY-COL-ITEM(67,3), TABLE-FILE-ARRAY-COL-ITEM(67,4).
MOVE 89 TO TABLE-FILE-ARRAY-COL-ITEM(67,5).

PERFORM R68 VARYING COLUMN-INDEX FROM 6 BY 1 UNTIL COLUMN-INDEX > 33.
MOVE 90 TO TABLE-FILE-ARRAY-COL-ITEM(68,1), TABLE-FILE-ARRAY-COL-ITEM(68,2).
MOVE 91 TO TABLE-FILE-ARRAY-COL-ITEM(68,3), TABLE-FILE-ARRAY-COL-ITEM(68,4).
MOVE 92 TO TABLE-FILE-ARRAY-COL-ITEM(68,5).

PERFORM R69 VARYING COLUMN-INDEX FROM 7 BY 1 UNTIL COLUMN-INDEX > 37.
MOVE 93 TO TABLE-FILE-ARRAY-COL-ITEM(69,1), TABLE-FILE-ARRAY-COL-ITEM(69,2).
MOVE 94 TO TABLE-FILE-ARRAY-COL-ITEM(69,3), TABLE-FILE-ARRAY-COL-ITEM(69,4).
MOVE 95 TO TABLE-FILE-ARRAY-COL-ITEM(69,5).

PERFORM R70 VARYING COLUMN-INDEX FROM 8 BY 1 UNTIL COLUMN-INDEX > 41.
MOVE 96 TO TABLE-FILE-ARRAY-COL-ITEM(70,1), TABLE-FILE-ARRAY-COL-ITEM(70,2).
MOVE 97 TO TABLE-FILE-ARRAY-COL-ITEM(70,3), TABLE-FILE-ARRAY-COL-ITEM(70,4).
MOVE 98 TO TABLE-FILE-ARRAY-COL-ITEM(70,5).

GO TO LOAD-852-STATES.

R60. MOVE 41 TO TABLE-FILE-ARRAY-COL-ITEM(60,COLUMN-INDEX).
R61. MOVE 45 TO TABLE-FILE-ARRAY-COL-ITEM(61,COLUMN-INDEX).
R62. MOVE 46 TO TABLE-FILE-ARRAY-COL-ITEM(62,COLUMN-INDEX).
R63. MOVE 47 TO TABLE-FILE-ARRAY-COL-ITEM(63,COLUMN-INDEX).
R64. MOVE 48 TO TABLE-FILE-ARRAY-COL-ITEM(64,COLUMN-INDEX).
R65. MOVE 49 TO TABLE-FILE-ARRAY-COL-ITEM(65,COLUMN-INDEX).
R66. MOV 69 TO TABLE-FILE-ARRAY-COL-ITEM(66, COLUMN-INDEX).
R67. MOV 68 TO TABLE-FILE-ARRAY-COL-ITEM(67, COLUMN-INDEX).
R68. MOV 70 TO TABLE-FILE-ARRAY-COL-ITEM(68, COLUMN-INDEX).
R69. MOV 71 TO TABLE-FILE-ARRAY-COL-ITEM(69, COLUMN-INDEX).
R70. MOV 72 TO TABLE-FILE-ARRAY-COL-ITEM(70, COLUMN-INDEX).
LOAD-651-652-STATES.
PERFORM R71 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 14.
MOVE 73 TO TABLE-FILE-ARRAY-COL-ITEM(71,3), TABLE-FILE-ARRAY-COL-ITEM(71,2).
PERFORM R72 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 23.
MOVE 0 TO TABLE-FILE-ARRAY-COL-ITEM(72,9), TABLE-FILE-ARRAY-COL-ITEM(72,12).
MOVE 74 TO TABLE-FILE-ARRAY-COL-ITEM(72,15), TABLE-FILE-ARRAY-COL-ITEM(72,19).
MOVE 75 TO TABLE-FILE-ARRAY-COL-ITEM(72,20), TABLE-FILE-ARRAY-COL-ITEM(72,24).
PERFORM R73 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 24.
MOVE 0 TO TABLE-FILE-ARRAY-COL-ITEM(73,9), TABLE-FILE-ARRAY-COL-ITEM(73,12).
MOVE 76 TO TABLE-FILE-ARRAY-COL-ITEM(73,15), TABLE-FILE-ARRAY-COL-ITEM(73,19).
PERFORM R74 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
GO TO LOAD-652-STATES.
R71. MOV 73 TO TABLE-FILE-ARRAY-COL-ITEM(71, COLUMN-INDEX).
R72. MOV 75 TO TABLE-FILE-ARRAY-COL-ITEM(72, COLUMN-INDEX).
R73. MOV 75 TO TABLE-FILE-ARRAY-COL-ITEM(73, COLUMN-INDEX).
R74. MOV 75 TO TABLE-FILE-ARRAY-COL-ITEM(74, COLUMN-INDEX).
LOAD-652-STATES.
MOVE 76 TO TABLE-FILE-ARRAY-COL-ITEM(75,25).
MOVE 77 TO TABLE-FILE-ARRAY-COL-ITEM(75,14).
MOVE 78 TO TABLE-FILE-ARRAY-COL-ITEM(75,28).
PERFORM R76 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 24.
MOVE 0 TO TABLE-FILE-ARRAY-COL-ITEM(76,9), TABLE-FILE-ARRAY-COL-ITEM(76,12).
MOVE 79 TO TABLE-FILE-ARRAY-COL-ITEM(76,15), TABLE-FILE-ARRAY-COL-ITEM(76,19).
PERFORM R77 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 23.
MOVE 0 TO TABLE-FILE-ARRAY-COL-ITEM(77,9), TABLE-FILE-ARRAY-COL-ITEM(77,12).
MOVE 79 TO TABLE-FILE-ARRAY-COL-ITEM(77,15), TABLE-FILE-ARRAY-COL-ITEM(77,19).
PERFORM R78 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 22.
MOVE 0 TO TABLE-FILE-ARRAY-COL-ITEM(78,9), TABLE-FILE-ARRAY-COL-ITEM(78,12).
MOVE 80 TO TABLE-FILE-ARRAY-COL-ITEM(78,19).
PERFORM R79 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
GO TO LOAD-652-652-STATES.
R76. MOV 79 TO TABLE-FILE-ARRAY-COL-ITEM(76, COLUMN-INDEX).
R77. MOV 80 TO TABLE-FILE-ARRAY-COL-ITEM(77, COLUMN-INDEX).
R78. MOV 79 TO TABLE-FILE-ARRAY-COL-ITEM(78, COLUMN-INDEX).
R79. MOVE B0 TO TABLE-FILE-ARRAY-COL-ITEM(79, COLUMN-INDEX).
LOAD-552-552-STATES.
MOVE B1 TO TABLE-FILE-ARRAY-COL-ITEM(80, 14).
MOVE B2 TO TABLE-FILE-ARRAY-COL-ITEM(80, 25), TABLE-FILE-ARRAY-COL-ITEM(80, 28).
PERFORM R81 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
MOVE B3 TO TABLE-FILE-ARRAY-COL-ITEM(82, 14).
MOVE B4 TO TABLE-FILE-ARRAY-COL-ITEM(82, 25), TABLE-FILE-ARRAY-COL-ITEM(82, 28).
PERFORM R83 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
PERFORM R84 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
PERFORM R85 VARYING COLUMN-INDEX FROM 1 BY 1 UNTIL COLUMN-INDEX > 29.
GO TO 0201-LOAD-STATE-TABLE-EXIT.
R81. MOVE B2 TO TABLE-FILE-ARRAY-COL-ITEM(81, COLUMN-INDEX).
R83. MOVE B4 TO TABLE-FILE-ARRAY-COL-ITEM(83, COLUMN-INDEX).
R84. MOVE B6 TO TABLE-FILE-ARRAY-COL-ITEM(84, COLUMN-INDEX).
R85. MOVE 55 TO TABLE-FILE-ARRAY-COL-ITEM(85, COLUMN-INDEX).
0201-LOAD-STATE-TABLE-EXIT.
EXIT.
******************************************************************************
* THIS PARAGRAPH IS CALLED FROM 0100 WHEN A RESPONSE OF 'ADMISSION' IS INPUT TO THE COMMAND 'INCON COMMAND'.
* THIS ROUTINE WILL PROMPT THE USER FOR ADMISSION DATES, USED TO INDICATE A RANGE. EACH DATE ENTERED
* WILL BE VERIFIED CORRECT AND A FINAL EDIT CHECK IS MADE TO ENSURE THE CORRECT CHRONOLOGICAL ORDER
* OF THE DATES. SHOULD A DATE BE ENTERED IN ERROR, A MESSAGE IS DISPLAYED INDICATING WHAT THE ERROR WAS
* AND PROMPTS AGAIN FOR THE INPUT. THIS ROUTINE IS REQUIRED FOR OPERATION.
0202-SELECT-ADM-DATES.
DISPLAY 'BEGINNING ADMISSION DATE(M.D.Y):' WITH NO ADVANCING.
ACCEPT GET-DATE.
UNSTRING GET-DATE DELIMITED BY ' ', OR ',' INTO HOLD-MONTH, HOLD-DAY, HOLD-YEAR.
* IF HOLD-YEAR EQUALS ZERO
* AND HOLD-MONTH EQUALS ZERO
* AND HOLD-DAY EQUALS ZERO
* MOVE LOW-VALUES TO BEGIN-ADMISSION-SELECT
* GO TO ADMISSION-SELECT-END.
IF HOLD-YEAR EQUALS ZERO
OR HOLD-MONTH EQUALS ZERO
OR HOLD-DAY EQUALS ZERO
DISPLAY 'ERROR: DATE'
DISPLAY 'REASON: WRONG DATE TYPED'
GO TO 0202-SELECT-ADM-DATES.
IF HOLD-MONTH > 12
DISPLAY 'ERROR: ADMISSION'
DISPLAY 'REASON: WRONG YEAR OR MONTH OR DAY'
GO TO 0202-SELECT-ADM-DATES.
IF HOLD-DAY > 31
DISPLAY 'ERROR: DATE'
DISPLAY 'REASON: WRONG DAY TYPED'
GO TO 0202-SELECT-ADM-DATES.
MOVE HOLD-DAY TO BEGIN-ADMISSION-SELECT.
ADMISSION-SELECT-END.
DISPLAY 'ENDING ADMISSION DATE(M.D.Y):' WITH NO ADVANCING.
ACCEPT GET-DATE.
UNSTRING GET-DATE DELIMITED BY ' ', OR ','
INTO HOLD-MONTH,HOLD-DAY,HOLD-YEAR.
 IF HOLD-YEAR EQUALS ZERO
 AND HOLD-MONTH EQUALS ZERO
 AND HOLD-DAY EQUALS ZERO
 MOVE HIGH-VALUES TO END-ADMISSION-SELECT
 GO TO 0202-SELECT-ADM-DATES-EXIT.
 IF HOLD-YEAR EQUALS ZERO
 OR HOLD-MONTH EQUALS ZERO
 OR HOLD-DAY EQUALS ZERO
 DISPLAY 'ERRORC:DATE'
 DISPLAY 'REASON=WRONG DATE TYPED'
 GO TO ADMISSION-SELECT-END.
 IF HOLD-MONTH > 12
 DISPLAY 'ERRORC:ADMISSION'
 DISPLAY 'REASON=WRONG YEAR OR MONTH OR DAY'
 GO TO ADMISSION-SELECT-END.
 IF HOLD-DAY > 31
 DISPLAY 'ERRORC:DAY'
 DISPLAY 'REASON=WRONG DAY TYPED'
 GO TO ADMISSION-SELECT-END.
 MOVE HOLD-DATE TO END-ADMISSION-SELECT.
 IF BEGIN-ADMISSION-SELECT > END-ADMISSION-SELECT
 DISPLAY 'ERRORC:SEQ'
 DISPLAY 'REASON=ENDING VALUE GREATER THAN BEGINNING VALUE'
 GO TO 0202-SELECT-ADM-DATES.
 0202-SELECT-ADM-DATES-EXIT.
 EXIT.
*****************************************************************************
*THIS PARAGRAPH IS CALLED FROM 0100 WHEN A RESPONSE OF 'DISCHARGE' IS INPUT TO THE COMMAND 'INCON COMMAND:'.
*THIS ROUTINE WILL PROMPT THE USER FOR DISCHARGE DATES USED TO INDICATE A RANGE. EACH DATE ENTERED WILL
*BE VERIFIED CORRECT AND A FINAL EDIT CHECK IS MADE TO ENSURE THE CORRECT CHRONOLOGICAL ORDER OF THE DATES.
*SHOULD A DATE BE ENTERED IN ERROR, A MESSAGE IS DISPLAYED INDICATING WHAT THE ERROR WAS AND PROMPTS
*AGAIN FOR THE INPUT. THIS ROUTINE IS REQUIRED FOR OPERATION.
0203-SELECT-DIS-DATES.
DISPLAY 'BEGINNING DISCHARGE DATE(M.D.Y):' WITH NO ADVANCING.
ACCEPT GET-DATE.
UNSTRING GET-DATE DELIMITED BY ',', OR ' '.
INTO HOLD-MONTH,HOLD-DAY,HOLD-YEAR.
 IF HOLD-YEAR EQUALS ZERO
 AND HOLD-MONTH EQUALS ZERO
 AND HOLD-DAY EQUALS ZERO
 MOVE LOW-VALUES TO BEGIN-DISCHARGE-SELECT
 GO TO DISCHARGE-SELECT-END.
 IF HOLD-YEAR EQUALS ZERO
 OR HOLD-MONTH EQUALS ZERO
 OR HOLD-DAY EQUALS ZERO
 DISPLAY 'ERRORC:DATE'
 DISPLAY 'REASON=WRONG DATE TYPED'
 GO TO 0203-SELECT-DIS-DATES.
 IF HOLD-MONTH > 12
 DISPLAY 'ERRORC:DISCHARGE'
 DISPLAY 'REASON=WRONG YEAR OR MONTH OR DAY'
GO TO 0203-SELECT-DIS-DATES.

IF HOLD-DAY > 31
  DISPLAY 'ERRORC:DAY'
  DISPLAY 'REASON=WRONG DAY TYPED'
  GO TO 0203-SELECT-DIS-DATES.
MOVE HOLD-DAY TO BEGIN-DISCHARGE-SELECT.
END-SELECT.
  DISPLAY 'ENDING DISCHARGE DATE(M,D,Y): ' WITH NO ADVANCING.
  ACCEPT GET-DATE.
  UNSTRING GET-DATE DELIMITED BY ',', OR ''
    INTO HOLD-MONTH,HOLD-DAY,HOLD-YEAR.
  IF HOLD-YEAR EQUALS ZERO
    AND HOLD-MONTH EQUALS ZERO
    AND HOLD-DAY EQUALS ZERO
      MOVE HIGH-VALUES TO END-DISCHARGE-SELECT
      GO TO 0203-SELECT-DIS-DATES-EXIT.
  IF HOLD-YEAR EQUALS ZERO
    OR HOLD-MONTH EQUALS ZERO
    OR HOLD-DAY EQUALS ZERO
      DISPLAY 'ERRORC:DATE'
      DISPLAY 'REASON=WRONG DATE TYPED'
      GO TO DISCHARGE-SELECT-EN.
  IF HOLD-MONTH > 12
    DISPLAY 'ERRORC:DISCHARGE'
    DISPLAY 'REASON=WRONG YEAR OR MONTH OR DAY'
    GO TO DISCHARGE-SELECT-EXIT.
  IF HOLD-DAY > 31
    DISPLAY 'ERRORC:DAY'
    DISPLAY 'REASON=WRONG DAY TYPED'
    GO TO DISCHARGE-SELECT-EXIT.
  MOVE HOLD-DAY TO END-DISCHARGE-SELECT.
  IF BEGIN-DISCHARGE-SELECT > END-DISCHARGE-SELECT
    DISPLAY 'ERRORC:SEQ'
    DISPLAY 'REASON=ENDING VALUE GREATER THAN BEGINNING VALUE'
    GO TO 0203-SELECT-DIS-DATES.
0203-SELECT-DIS-DATES-EXIT.
EXIT.

**THIS PARAGRAPH IS INVOKED FROM 0100 WHEN A RESPONSE OF 'LIST' IS INPUT TO THE COMMAND 'INCON COMMAND':**

**THIS ROUTINE WILL DISPLAY THE STATE TABLE AS IT IS CURRENTLY BEING USED. THIS ROUTINE IS NOT REQUIRED FOR NORMAL OPERATION.**

0204-DISPLAY-STATE-TABLE.

MOVE 1 TO COLUMN-INDEX, ROW-INDEX.
DISPLAY ' '.
DISPLAY 'EXISTING STATE TABLE'.
DISPLAY '=' 'WITH NO ADVANCING.'.
COLUMN-LABEL-LOOP.
MOVE COLUMN-INDEX TO STATE-DISPLAY.
DISPLAY ' ',STATE-DISPLAY WITH NO ADVANCING.
ADD 1 TO COLUMN-INDEX.
IF COLUMN-INDEX < 30
GO TO COLUMN-LABEL-LOOP.
DISPLAY ' ',
MOVE 1 TO COLUMN-INDEX.
DISPLAY ' ' WITH NO ADVANCING.
COLUMN-LINE-LOOP.
DISPLAY '-----' WITH NO ADVANCING.
ADD 1 TO COLUMN-INDEX.
IF COLUMN-INDEX < 30
GO TO COLUMN-LINE-LOOP.
DISPLAY ' ',
MOVE 1 TO COLUMN-INDEX.
NEW-ROW-LOOP.
IF ROW-INDEX > EXIT-STATE
GO TO 0204-DISPLAY-STATE-TABLE-EXIT.
MOVE ROW-INDEX TO STATE-DISPLAY.
DISPLAY STATE-DISPLAY, '! ' WITH NO ADVANCING.
ROW-VALUE-LOOP.
MOVE TABLE-FILE-ARRAY-CQL-ITEM(ROW-INDEX, COLUMN-INDEX) TO STATE-DISPLAY.
DISPLAY STATE-DISPLAY, ' ' WITH NO ADVANCING.
ADD 1 TO COLUMN-INDEX.
IF COLUMN-INDEX > 29
ADD 1 TO ROW-INDEX
MOVE 1 TO COLUMN-INDEX
DISPLAY ' '
GO TO NEW-ROW-LOOP.
GO TO ROW-VALUE-LOOP.
0204-DISPLAY-STATE-TABLE-EXIT.
EXIT.
******************************************************************************
*THIS ROUTINE IS INVOKED FROM 0100 WHEN A RESPONSE OF 'PPN' IS INPUT TO THE COMMAND 'INCON COMMAND:'.
*THE CODE WILL ALLOW UP TO 10 5 DIGIT PROGRAMMER NUMBERS BETWEEN THE VALUES 77316 AND 77336. AN INPUT OF BLANK
*WILL EXIT THIS ROUTINE. THIS ROUTINE IS NOT REQUIRED FOR NORMAL RUNS, AS THE SYSTEM WILL DEFAULT TO THE
*LIST OF ALL CURRENT VALID PROGRAMMER NUMBERS.
0205-LOAD-PPNS.
MOVE 'YES' TO PPN-SELECT.
MOVE ZEROS TO PPN-COUNTER.
PPN-LOAD-ACCEPT.
DISPLAY 'PROGRAMMER NUMBER = ', WITH NO ADVANCING.
ACCEPT PROG-NUMBER.
IF (PROG-NUMBER EQUALS ZERO) AND (PPN-COUNTER EQUALS ZERO)
MOVE SPACES TO PPN-SELECT
GO TO 0205-LOAD-PPNS-EXIT.
IF (PROG-NUMBER = 0) AND (PPN-COUNTER = 0)
MOVE 'NO' TO PPN-SELECT
MOVE 10 TO PPN-NUMBER
GO TO 0205-LOAD-PPNS-EXIT.
IF PROG-NUMBER EQUALS ZERO
GO TO PPN-LOAD-FINISH.
IF PROG-NUMBER < 77316
OR PROG-NUMBER > 77336
DISPLAY 'ERROR:PPN'
DISPLAY 'REASON: WRONG PROGRAMMER NUMBER TYPED'
GO TO PPN-LOAD-ACCEPT.
ADD 1 TO PPN-COUNTER.
MOVE PROG-NUMBER TO PPN-NUMBER-ITEM(PPN-COUNTER).
GO TO PPN-LOAD-ACCEPT.
PPN-LOAD-FINISH.
MOVE PPN-COUNTER TO PPN-NUMBER.
0205-LOAD-PPNS-EXIT.
EXIT.

* THIS PARAGRAPH IS CALLED FROM 0100 WHEN A RESPONSE OF 'DOCUMENTATION' IS INPUT TO THE COMMAND
* 'INCON COMMAND'. THE ROUTINE SIMPLY DISPLAYS TEXT INDICATING THE USE OF THE SYSTEM, HOW IT PERFORMS ITS
* FUNCTIONS, THE DISTRIBUTION OF THE OUTPUT, AND WHERE TO REFER TO FOR PROGRAMMING SPECIFICS. THIS ROUTINE
* IS NOT REQUIRED FOR NORMAL RUNS.
0206-DISPLAY-DOC.
DISPLAY 'THIS SYSTEM IS DESIGNED TO INTERFACE WITH THE ADES DATA COLLECTION'.
DISPLAY 'SYSTEM. ISAM FILES CONTAINING CLIENT INFORMATION ARE RELATED TO OTHER'.
DISPLAY 'ISAM FILES CONTAINING CLIENT SPECIFIC DATA, INCLUDING POINTERS FOR'.
DISPLAY 'DIRECT ACCESS TO THE INFORMATION FILES. THE LATTER FILES CONTAIN KEYS',
DISPLAY 'WHICH ALLOW FOR DIRECT LOOKUP FOR EACH CLIENT, FOR FURTHER FILE'.
DISPLAY 'RELATIONSHIPS, PLEASE CONSULT THE ADES SYSTEM DOCUMENTATION MANUAL'.
DISPLAY 'BECAUSE OF THESE FILE RELATIONSHIPS, MINIMAL INPUT IS NECESSARY, AS A'.
DISPLAY 'CLIENT IS EITHER REJECTED OR PASSED THROUGH FOR ANALYSIS BASED ON'.
DISPLAY 'ADMISSION AND DISCHARGE DATES FOUND AFTER THE FIRST DIRECT READ'.
DISPLAY 'VARIOUS FILES WILL BE DONE IF AND ONLY IF THE CLIENT SPECIFIC DATA'.
DISPLAY 'FILE CONTAINS A POINTER TO THE PARTICULAR INFORMATION FILE. THUS EACH'.
DISPLAY 'CLIENT SPECIFIC DATA RECORD IS READ ONCE, AND THE POTENTIAL EXISTS FOR'.
DISPLAY 'UP TO FOUR ADDITIONAL READS PER CLIENT'.
DISPLAY 'THE INTERFACE CONSISTENCY PROJECT WILL EXAMINE ALL CLIENT INFORMATION'.
DISPLAY 'AVAILABLE. SHOULD A CLIENT NOT HAVE ONE OR MORE OF THE FOUR FORMS'.
DISPLAY 'REQUARED FOR NORMAL PROCESSING, THE INCONSISTENCY EVALUATION PER'.
DISPLAY 'FORMED BY THE SYSTEM WILL NOT RECORD ANY ERRORS, BUT RATHER DROP'.
DISPLAY 'THROUGH THAT PARTICULAR DATA SET. THIS ENSURES ALL ERRORS RECORDED'.
DISPLAY 'ARE DUE TO INCORRECT DATA COLLECTION OR ENTRY AND NOT THE RESULT OF'.
DISPLAY 'MISSING CLIENT DATA'.
DISPLAY 'THE METHOD OF VERIFYING CONSISTENT DATA WILL BE TABLE-LOOKUP. THIS'.
DISPLAY 'PROCESS WILL CLOSELY PARALLEL THE SYNTACTICAL ANALYSIS PERFORMED IN'.
DISPLAY 'COMPILERS, USING LEXICAL ANALYSIS TECHNIQUES. THIS PROCESS SHOULD'.
DISPLAY 'MINIMIZE THE TIME REQUIRED TO EXAMINE DATA ONCE IT HAS BEEN INPUT TO'.
DISPLAY 'THE SOFTWARE'.
DISPLAY 'ALL CLIENT INFORMATION WILL BE READ INTO THE SYSTEM AND THE ELEMENTS'.

163
TO BE ANALYZED WILL BE MOVED INTO A STRING FOR USE IN TABLE TRAVERSAL.

THIS PROCESS OF CREATING A LIST OF ALL ELEMENTS TO BE EXAMINED
CIRCUMVENTS THE NEED FOR A COMPLICATED AND CUMBERSOME FILTERING ALGORITHM.

THE STATE TABLE TO BE USED WILL BE A DETERMINISTIC FINITE AUTOMATA.

PROVEN TECHNIQUES WILL BE USED TO MINIMIZE THE TABLE ONCE INITIAL DESIGN IS COMPLETE. ALL INCOMING ELEMENTS WILL BE TRANSLATED TO AN INDEX BASED ON DEFINITIONS DEVISED. THIS INFORMATION WILL BECOME PART OF THE SYSTEM DOCUMENTATION NOTEBOOK (SDN). THIS DOCUMENT WILL CONTAIN ALL INFORMATION NECESSARY TO UNDERSTAND, TROUBLESHOOT, AND UPDATE THE SYSTEM.

THE OUTPUT SHALL BE WRITTEN TO DISK TAKING FULL ADVANTAGE OF THE SCRATCHPAD FACILITY CURRENTLY AVAILABLE ON THE EXISTING COMPUTER.

SYSTEM. THE MEMOS IN THIS FILE ARE PRECEDED BY A PAGE-TOP FLAG FOR LINE PRINTER OUTPUT. THE LOCAL PROGRAM TPRINT WILL BE USED TO PRODUCE ON-LINE MEMOS FOR DISTRIBUTION TO THE COORDINATING AGENCIES. THE LINE PRINTER COPY WILL BE MAINTAINED AT THE ADES OFFICE FOR HISTORICAL PURPOSES.

THIS PARAGRAPH IS CALLED FROM 0100 WHEN A RESPONSE OF 'HELP' IS INPUT TO THE COMMAND 'INCON COMMAND:'. A LIST OF ALL AVAILABLE COMMANDS IS DISPLAYED WITH A SHORT EXPLANATION OF THE FUNCTION PERFORMED FOR EACH COMMAND. THIS ROUTINE IS NOT REQUIRED FOR NORMAL RUNS.

DISPLAY 'TYPE FOR RESPONSE'.
DISPLAY '----- ---'.
DISPLAY 'ADM'.
DISPLAY 'ADVANCE'.
DISPLAY 'DID'.
DISPLAY 'PPN'.
DISPLAY 'PP'.
DISPLAY 'PRINT'.
DISPLAY 'INCON'.
DISPLAY 'LST'.
DISPLAY 'DOC'.
DISPLAY 'HELP'.
DISPLAY '<RETURN>'.

0206-DISPLAY-DOC-EXIT.
EXIT.

******************************************************************************************************************
*THIS ROUTINE, CALLED FROM 0100, IS THE SECOND CONTROL LEVEL FOR THE SYSTEM. IT IS INVOKED BY A RESPONSE OF *
* 'INCONSISTENCY' TO THE COMMAND 'INCON COMMAND:'. THE FIRST SECTION OF THIS ROUTINE INITIALIZES ALL REPORT
*VARIABLES WHICH ARE GENERIC TO EACH OUTPUT REPORT TO BE GENERATED.* THE PARAGRAPH THEN PROMPTS FOR A DECISION
*ON SELECTING ALL PROGRAMS FROM EACH AGENCY. THIS VALUE WILL BE USED IN THE CALL TO PARAGRAPH 0301-SELECT-PROGRAMS.*
*THE PPN-TABLE IS THEN LOADED BASED ON THE OPTION OF PPN SELECTION. TWO MACRO SYSTEM INTERNAL ROUTINES (TTYWID -
*FILEPP) ARE INVOKED HERE. THE FIRST SETTING THE NECESSARY TERMINAL WIDTH AND THE LATTER GAINING ACCESS TO
*THE DISK AREA ASSIGNED TO THE GIVEN PPN. THIS CONTROL ROUTINE WILL LOOP THROUGH THE ENTIRE LIST OF
*PROGRAMMER NUMBERS BEFORE RELENSUHING CONTROL TO THE NEXT LEVEL. THE FOLLOWING LIST DENOTES THOSE ROUTINES
*CALLED FROM THIS PARAGRAPH:
 *
0208-SELECT-AGENCIES,
OPEN OUTPUT OUTPUT-FILE.
MOVE TODAY TO TODAYS-DATE.
MOVE JULIAN-MONTH(TODAYS-MONTH) TO M5-1.
MOVE TODAYS-DAY TO M5-2.
MOVE TODAYS-YEAR TO M5-3.
IF BEGIN-ADMISSION-SELECT < BEGIN-DISCHARGE-SELECT
MOVE BEGIN-ADM-MONTH TO M6-2
MOVE BEGIN-ADM-DAY TO M6-3
MOVE BEGIN-ADM-YEAR TO M6-4
ELSE
MOVE BEGIN-DIS-MONTH TO M6-2
MOVE BEGIN-DIS-DAY TO M6-3
MOVE BEGIN-DIS-YEAR TO M6-4.
ENDIF-ADMISSION-SELECT > END-DISCHARGE-SELECT
MOVE END-ADM-MONTH TO M6-5
MOVE END-ADM-DAY TO M6-6
MOVE END-ADM-YEAR TO M6-7
ELSE
MOVE END-DIS-MONTH TO M6-5
MOVE END-DIS-DAY TO M6-6
MOVE END-DIS-YEAR TO M6-7.
ENDIF(END-ADMISSION-SELECT EQUALS HIGH-VALUES)
OR(END-DISCHARGE-SELECT EQUALS HIGH-VALUES)
MOVE TODAYS-MONTH TO M6-5
MOVE TODAYS-DAY TO M6-6
MOVE TODAYS-YEAR TO M6-7.
ENTER MACRO TTYWID USING 125.
MOVE 1 TO PPN-COUNTER.
ALL-PGMS-LOOP.
DISPLAY '/'.
DISPLAY 'ARE ALL PGMS FOR ALL AGENCIES VALID ? ', WITH NO ADVANCING.
ACCEPT ALL-PGMS.
IF ALL-PGMS EQUAL SPACES
GO TO SELECT-AGENCIES-FINISH.
IF (ALL-PGMS NOT EQUAL 'YE')
AND (ALL-PGMS NOT EQUAL 'NO')
DISPLAY 'ERROR: INVALID RESPONSE TYPED'
GO TO ALL-PGMS-LOOP.
SELECT-AGENCIES-LOOP.
IF PPN-COUNTER > PPN-NUMBER
    GO TO SELECT-AGENCIES-FINISH.
IF PPN-SELECT EQUALS 'YES'
    MOVE PPN-NUMBER-ITEM(PPN-COUNTER) TO PROG-NUMBER
ELSE
    MOVE ALL-PPN-NUMBER-ITEM(PPN-COUNTER) TO PROG-NUMBER.
ENTER MACRO FILEPP USING PROJ-NUMBER,PROG-NUMBER,WHOLE-NUMBER.
PERFORM 0301-SELECT-PROGRAMS THRU 0301-SELECT-PROGRAMS-EXIT.
PERFORM 0302-WRITE-MEMO-HEADER THRU 0302-WRITE-MEMO-HEADER-EXIT.
PERFORM 0303-GET-DATA-BY-PROGRAM THRU 0303-GET-DATA-BY-PROGRAM-EXIT.
ADD 1 TO PPN-COUNTER.
GO TO SELECT-AGENCIES-LOOP.
SELECT-AGENCIES-FINISH.
CLOSE OUTPUT-FILE.
DISPLAY ' '.
DISPLAY 'END OF INTERFORM CONSISTENCY RUN.'.
DISPLAY ' '.
DISPLAY ' '.
0208-SELECT-AGENCIES-EXIT.
EXIT.
******************************************************************************
* LEVEL 3
******************************************************************************
* THIS ROUTINE, CALLED FROM 0208 WILL FIRST OBTAIN THE AGENCY NAME AND PLACE IT IN THE PROPER OUTPUT LOCATION.
* THE SUB-P ARAPHRAGH SELECT-OPTION WILL ALLOW FOR PROGRAM SELECTION BASED ON THE VALUE OF ALL-PPNS SET IN
* PARAGRAPH 0208. THIS SUB- PARAPHRAGH WILL ONLY ALLOW FOUR RESPONSES; YES, INDICATING A DESIRE TO SELECT
* PROGRAMS; NO, INDICATING ALL PROGRAMS ARE VALID; HELP, REQUESTING FURTHER INFORMATION; AND: <BLANK>, INDICATING
* A RETURN TO THE PREVIOUS LEVEL OF CONTROL. TO SELECT PROGRAMS, THE ROUTINE 0401-SELECT-ALL-PROGRAMS IS
* CALLED, AND THE HELP REQUEST INVOKES 0402-DPY-PGM-SEL-HELP. WHEN ALL PROGRAMS ARE VALID, THE SUB-
* PARAPHRAGH ALL-PROGRAM-READ IS ACTUATED, PLACING ALL PROGRAM DATA IN THE APPROPRIATE ARRAY. THE FOLLOWING
* LIST DENOTES THOSE ROUTINES CALLED FROM THIS PARAPHRAGH:
* 0401-SELECT-ALL-PROGRAMS (CONDITIONAL)
* 0402-DPY-PGM-SEL-HELP (CONDITIONAL)
0301-SELECT-PROGRAMS.
OPEN INPUT PROGRAM-FILE.
MOVE 'CA' TO PROGRAM-SEARCH-ID.
READ PROGRAM-FILE INVALID
CLOSE PROGRAM-FILE
MOVE '0301-1' TO READ-ERROR-LOC
MOVE 'NO CA FOUND' TO READ-ERROR-DESC
MOVE PROGRAM-SEARCH-KEY TO READ-ERROR-KEY
PERFORM 0701-READ-ERROR-DUMP THRU 0701-READ-ERROR-DUMP-EXIT
STOP RUN.
MOVE PROGRAM-NAME TO MS-1.
MOVE PROG-NUMBER TO PROG-DISPLAY.
MOVE ZEROS TO PROGRAM-STACK-COUNTER.
MOVE SPACES TO PROGRAM-STACK-TABLE.
SELECT-OPTION.
IF ALL-PPNS = 'YE'
    GO TO SELECT-OPTION-FIN.
DISPLAY ' '.
DISPLAY 'CA:' ',PROGRAM-NAME,' PPN= ',PROG-DISPLAY.
DISPLAY 'WOULD YOU LIKE TO SELECT PROGRAMS? ', WITH NO ADVANCING.
ACCEPT PROGRAM-SELECT-OPTION.
IF (PROGRAM-SELECT-OPTION NOT EQUAL SPACES)
AND (PROGRAM-SELECT-OPTION NOT EQUAL 'YE')
AND (PROGRAM-SELECT-OPTION NOT EQUAL 'NO')
AND (PROGRAM-SELECT-OPTION NOT EQUAL 'HE')
DISPLAY 'ERROR*: WRONG OPTION TYPED'
GO TO SELECT-OPTION.
SELECT-OPTION-FIN.
IF PROGRAM-SELECT-OPTION EQUALS 'YE'
PERFORM 0401-SELECT-ALL-PROGRAMS THRU
0401-SELECT-ALL-PROGRAMS-EXIT
GO TO 0301-SELECT-PROGRAMS-EXIT.
IF PROGRAM-SELECT-OPTION EQUALS 'HE'
PERFORM 0402-DPY-PGM-SEL-HELP THRU 0402-DPY-PGM-SEL-HELP-EXIT
GO TO SELECT-OPTION.
CLOSE PROGRAM-FILE.
OPEN INPUT PROGRAM-FILE.
MOVE LOW-VALUES TO PROGRAM-SEARCH-KEY.
ALL-PROGRAM-READ.
READ PROGRAM-FILE INVALID
CLOSE PROGRAM-FILE
GO TO 0301-SELECT-PROGRAMS-EXIT.
IF PROGRAM-RECORD-KEY IS NOT NUMERIC
CLOSE PROGRAM-FILE
GO TO 0301-SELECT-PROGRAMS-EXIT.
ADD 1 TO PROGRAM-STACK-COUNTER.
MOVE PROGRAM-RECORD-KEY TO PROGRAM-STACK-1D(PROGRAM-STACK-COUNTER).
MOVE PROGRAM-NAME TO PROGRAM-STACK-NAME(PROGRAM-STACK-COUNTER).
GO TO ALL-PROGRAM-READ.
0301-SELECT-PROGRAMS-EXIT.
EXIT:
******************************************************************************
*THIS ROUTINE, CALLED FROM 0208, WILL DISPLAY THE TIME AND AGENCY THAT IS CURRENTLY BEING EXAMINED. THIS*
*INFORMATION IS VITAL TO DETERMINE RUNTIME EFFICIENCY AND EXPECTED RUN DURATION. THE SUB-PARAGRAPH WRITE-MEMO*
*WILL WRITE THE MEMO HEADER PREVIOUSLY LOADED TO THE OUTPUT FILE.*
0302-WRITE-MEMO-HEADER.
IF (PPN-COUNTER = 1)
DISPLAY '*
DISPLAY 'ADVANCE PAPER TO BURST-LINE THEN TYPE RETURN'
ACCEPT YES-NO.
MOVE TODAY TO TIME-DISPLAY.
DISPLAY '/.
DISPLAY 'TIME: ', HOURS:1,':', MINUTES:1,':', SECONDS.
DISPLAY '/.
DISPLAY 'CA: ', PROGRAM-NAME, ' PPN= ', PROG-DISPLAY.
DISPLAY '/.
DISPLAY '.
MOVE SPACES TO OUTPUT-RECORD.
WRITE OUTPUT-RECORD AFTER ADVANCING TOP-OF-PAGE.
PERFORM DISPLAY-AND-WRITE-MEMO
VARYING MEMO-COUNTER FROM 1 BY 1
UNTIL MEMO-COUNTER > 21.
GO TO 0302-WRITE-MEMO-HEADER-EXIT.
DISPLAY-AND-WRITE-MEMO.
MOVE MEMO-TABLE-ITEM(MEMO-COUNTER) TO OUTPUT-RECORD.
WRITE OUTPUT-RECORD.
0302-WRITE-MEMO-HEADER-EXIT.
EXIT.

*THIS PARAGRAPH, CALLED FROM 0208, IS THE THIRD MAJOR CONTROL LEVEL, INITIALIZING THOSE VARIABLES
NECESSARY WHENEVER A NEW PROGRAM DATA SET IS REQUIRED. CONTROL IS PASSED ON TO THE NEXT LEVEL FOR
DEALING WITH SPECIFIC CLIENT DATA. WHEN THE LIST OF PROGRAMS IS EXHAUSTED, THE OUTPUT MEMO RECORD IS
COMPLETED, BASED ON THE QUANTITY OF ERRORS ENCOUNTERED. THE FOLLOWING LIST DENOTES THOSE ROUTINES
INVOKED FROM THIS PARAGRAPH:

* 0403-GET-CLIENT-DATA
* 0404-FINISH-CURRENT-MEMO
* 0405-DISPLAY-PGM-TOTALS

0303-GET-DATA-BY-PROGRAM.
MOVE ZERO TO PROGRAM-COUNTER.
OPEN INPUT CLIENT-DATA-FILE.
MOVE 0 TO ERROR-COUNT.

GET-DATA-BY-PROGRAM-CLIENT.
MOVE LOW-VALUES TO CLIENT-DATA-SEARCH-KEY.
ADD 1 TO PROGRAM-COUNTER.
IF PROGRAM-COUNTER > PROGRAM-STACK-COUNTER
GO TO GET-DATA-BY-PROGRAM-FINISH.

DISPLAY 'P ID : PROGRAM-STACK-ID(PROGRAM-COUNTER), ' , PROGRAM-STACK-NAME(PROGRAM-COUNTER).
MOVE PROGRAM-STACK-ID(PROGRAM-COUNTER) TO CLIENT-DATA-SEARCH PROGRAM, MEMQ-PID,TOTALS-ARRAY-ID(PROGRAM-COUNTER).
READ CLIENT-DATA-FILE INVALID
MOVE LOW-VALUES TO CLIENT-DATA-SEARCH-KEY.
PERFORM 0403-GET-CLIENT-DATA THRU 0403-GET-CLIENT-DATA-EXIT.
IF CLIENT-MSG = 1
MOVE SPACES TO OUTPUT-RECORD
WRITE OUTPUT-RECORD
MOVE 0 TO CLIENT-MSG.
GO TO GET-DATA-BY-PROGRAM-CLIENT.

GET-DATA-BY-PROGRAM-FINISH.
IF ERROR-COUNT = 0
PERFORM 0404-FINISH-CURRENT-MEMO THRU 0404-FINISH-CURRENT-MEMO-EXIT
VARYING MEMO-COUNTER FROM 30 BY 1
UNTIL MEMO-COUNTER > 32.
PERFORM 0404-FINISH-CURRENT-MEMO THRU 0404-FINISH-CURRENT-MEMO-EXIT
VARYING MEMO-COUNTER FROM 23 BY 1
UNTIL MEMO-COUNTER > 26.
PERFORM 0405-DISPLAY-PGM-TOTALS THRU 0405-DISPLAY-PGM-TOTALS-EXIT
VARYING DISPLAY-COUNTER FROM 1 BY 2
UNTIL TOTALS-ARRAY-ID(DISPLAY-COUNTER) EQUALS SPACES.
MOVE SPACES TO TOTALS-ARRAY.
IF ERROR-COUNT > 0
PERFORM 0404-FINISH-CURRENT-MEMO THRU 0404-FINISH-CURRENT-MEMO-EXIT
VARYING MEMO-COUNTER FROM 26 BY 1
UNTIL MEMO-COUNTER > 29
ELSE
PERFORM 0404-FINISH-CURRENT-MEMO THRU 0404-FINISH-CURRENT-MEMO-EXIT
VARING MEMO-COUNTER FROM 32 BY 1
UNTIL MEMO-COUNTER > 35.
0303-GET-DATA-BY-PROGRAM-EXIT.
EXIT.
******************************************************************************
* LEVEL 4
******************************************************************************
*THIS PARAGRAPH IS CALLED FROM 0301 TO ALLOW THE USER TO SELECT THOSE AGENCY PROGRAMS DESIRED. THE *
*ROUTINE SEQUENTIALLY READS THE PROGRAM-DATA-FILE, DISPLAYING THE APPLICABLE INFORMATION AND REQUIRES A *
*RESPONSE OF 'YES' TO ENABLE THE PROGRAM FOR THE REPORT.
0401-SELECT-ALL-PROGRAMS.
CLOSE PROGRAM-FILE.
OPEN input program-file.
MOVE low-values TO program-search-key.
ALL-SELECT-PROGRAM-READ.
read program-file invalid
CLOSE PROGRAM-FILE
GO TO 0401-SELECT-ALL-PROGRAMS-EXIT.
IF program-record-key is NOT numeric
CLOSE PROGRAM-FILE
GO TO 0401-SELECT-ALL-PROGRAMS-EXIT.
DISPLAY ".
DISPLAY 'PROGRAM NAME:', PROGRAM NAME.
DISPLAY 'PROGRAM LIC': PROGRAM LICENSE-_NUM 'PROGRAM ID':
PROGRAM-RECORD-KEY ? WITH NO ADVANCING.
ACCEPT YES-NO.
IF YES-NO EQUAL TO 'YES'
ADD 1 TO PROGRAM-STACK-COUNTER
MOVE program-record-key TO PROGRAM-STACK-ID(PROGRAM-STACK-COUNTER)
MOVE program-name TO PROGRAM-STACK-NAME(PROGRAM-STACK-COUNTER).
GO TO ALL-SELECT-PROGRAM-READ.
0401-SELECT-ALL-PROGRAMS-EXIT.
EXIT.
******************************************************************************
*THIS PARAGRAPH, CALLED FROM 0301, WILL DISPLAY TEXT EXPLAINING THE COMMANDS AVAILABLE AT THIS LEVEL *
*OF LOGIC.
0402-DPY-PM-SM-SEL-HELP.
DISPLAY ",
DISPLAY ",
DISPLAY 'TYPE FOR RESPONSE'.
DISPLAY '===>' INDICATING YOUR DESIRE TO CHECK INTERFORM CONSISTENCY'.
DISPLAY '===>' ON ONLY A SELECT SUBSET OF PROGRAMS. EACH VALID PROGRAM',
DISPLAY '===>' WILL BE DISPLAYED. SHOULD YOU WANT A REPORT ON THAT PROGRAM, '
DISPLAY '===>' ENTER QUOTE YES QUOTE'.
DISPLAY '===>' INDICATING YOUR DESIRE TO CHECK INTERFORM CONSISTENCY'.
DISPLAY '===>' ON ALL PROGRAMS'.
DISPLAY 'HELP' TYPE THIS LIST'.

169
**DISPLAY** **:**
**DISPLAY** **:**
0402-DPY-PGM-SEL-HELP-EXIT.
**EXIT**

************************************************
**THIS ROUTINE IS CALLED FROM 0303 TO LOOP THROUGH THE CLIENT DATA FILES, VERIFY THE PROGRAM ID, AND
**PASS CONTROL TO THE NEXT LEVEL OF CODE. THIS SECTION OF CODE WILL WRITE A BLANK LINE IN THE OUTPUT MEMO.
**EVERY TIME A NEW CLIENT ID IS ENCOUNTERED IF AND ONLY IF THERE WAS OUTPUT FROM THE PREVIOUS CLIENT EXAMINED.
**THIS PARAGRAPH IS ALSO USED TO VERIFY THAT THE CURRENT CLIENT DATA MEETS THE CRITERIA SET PREVIOUSLY.
**IF THE DATA MATCHES THE CLIENT FORM DATA IS THEN LOADED AND CONTROL IS PASSED TO THE NEXT LEVEL. THE
**FOLLOWING LIST DENOTES THOSE ROUTINES INVOKED FROM THIS PARAGRAPH:
**
**0501-SEARCH-LOAD-551
**0502-SEARCH-LOAD-552
**0503-SEARCH-LOAD-651
**0504-SEARCH-LOAD-652
**0505-STATE-LOOKUP
**
0403-GET-CLIENT-DATA.
**READ CLIENT-DATA-FILE INVALID
**GO TO 0402-GET-CLIENT-DATA-EXIT.
**IF (CLIENT-DATA-PROGRAM NOT EQUAL PROGRAM-STACK-ID(PROGRAM-COUNTER))
**PERFORM SKIP-LINE-BETWEEN-PGMS
**GO TO 0403-GET-CLIENT-DATA-EXIT.
**IF CLIENT-DATA-ADMISSION IS NOT LESS THAN BEGIN-ADMISSION-SELECT
**AND CLIENT-DATA-ADMISSION IS NOT GREATER THAN END-ADMISSION-SELECT
**GO TO GET-CLIENT-DATA-PASS.
**IF CLIENT-DATA-PASS IS NOT LESS THAN BEGIN-PASS-SELECT
**AND CLIENT-DATA-PASS IS NOT GREATER THAN END-PASS-SELECT
**GO TO GET-CLIENT-DATA-PASS.
**GO TO CHECK-FOR-MESSAGE.
**
**SKIP-LINE-BETWEEN-PGMS.
**IF (PROGRAM-STACK-COUNTER > 1) AND (PROGRAM-ERRORS > 0)
**MOVE SPACES TO OUTPUT-RECORD
**WRITE OUTPUT-RECORD
**MOVE 0 TO PROGRAM-ERRORS.
**
**GET-CLIENT-DATA-PASS.
**MOVE CLIENT-DATA-ID TO MEMO-CID.
**MOVE CLIENT-DATA-SERVICE TO MEMO-SC.
**ADD 1 TO TOTALS-ARRAY-COUNT(PROGRAM-COUNTER).
**MOVE SPACES TO TEST-1-RECORD.
**PERFORM 0501-SEARCH-LOAD-551 THRU 0501-SEARCH-LOAD-551-EXIT.
**PERFORM 0502-SEARCH-LOAD-552 THRU 0502-SEARCH-LOAD-552-EXIT.
**PERFORM 0503-SEARCH-LOAD-651 THRU 0503-SEARCH-LOAD-651-EXIT.
**PERFORM 0504-SEARCH-LOAD-652 THRU 0504-SEARCH-LOAD-652-EXIT.
**PERFORM 0505-STATE-LOOKUP THRU 0505-STATE-LOOKUP-EXIT.
**
**CHECK-FOR-MESSAGE.
**IF CLIENT-MSG = 1
**MOVE SPACES TO OUTPUT-RECORD
**WRITE OUTPUT-RECORD
**MOVE 0 TO CLIENT-MSG.
**GO TO 0403-GET-CLIENT-DATA.
**
0403-GET-CLIENT-DATA-EXIT.
**EXIT.
THIS SMALL PARAGRAPH RECEIVES SEVERAL CALLS FROM 0303 TO SIMPLY MOVE MEMO ARRAY ELEMENTS TO THE
OUTPUT FILE AND WRITE EACH RECORD. THE CONTROL VARIABLES (ARRAY INDEXES AND LOOP VARIABLES) ARE
DETERMINED BY EACH CALLING STATEMENT.

0404-FINISH-CURRENT-MEMO.
MOVE MEMO-TABLE-ITEM(MEMO-COUNTER) TO OUTPUT-RECORD.
WRITE OUTPUT-RECORD AFTER ADVANCING 1 LINE.
0404-FINISH-CURRENT-MEMO-EXIT.
EXIT.

THIS ROUTINE, CALLED FROM 0303, WILL DISPLAY TOTAL COUNTS OF CLIENT DATA FORMS EXAMINED FOR EACH
PROGRAM ON THE REPORT. THE CODE WILL PLACE THREE PROGRAMS RESULTS ON ONE LINE OF OUTPUT BEFORE
WRITING TO DISK, ALL MEMO VARIABLES ARE RE-INITIALIZED AFTER A WRITE.

0405-DISPLAY-PGM-TOTALS.
IF DISPLAY-COUNTER NOT EQUAL 1
ADD 1 TO DISPLAY-COUNTER.
IF TOTALS-ARRAY-ID(DISPLAY-COUNTER) EQUALS SPACES
GO TO 0405-DISPLAY-PGM-TOTALS-EXIT.
MOVE ZERO TO CLIENTS-PROCESSED-1, CLIENTS-PROCESSED-2, CLIENTS-PROCESSED-3.
MOVE TOTALS-ARRAY-ID(DISPLAY-COUNTER) TO PROCESS-ID-1.
MOVE TOTALS-ARRAY-COUNT(DISPLAY-COUNTER) TO CLIENTS-PROCESSED-1.
MOVE TOTALS-ARRAY-ID(DISPLAY-COUNTER + 1) TO PROCESS-ID-2.
MOVE TOTALS-ARRAY-COUNT(DISPLAY-COUNTER + 1) TO CLIENTS-PROCESSED-2.
MOVE TOTALS-ARRAY-ID(DISPLAY-COUNTER + 2) TO PROCESS-ID-3.
MOVE TOTALS-ARRAY-COUNT(DISPLAY-COUNTER + 2) TO CLIENTS-PROCESSED-3.
IF TOTALS-ARRAY-ID(DISPLAY-COUNTER + 2) NOT EQUAL SPACES
MOVE TOTAL-OUTPUT-LINE TO OUTPUT-RECORD
GO TO DISPLAY-PROGRAM-TOTALS-FINISH.
IF TOTALS-ARRAY-ID(DISPLAY-COUNTER + 1) EQUALS SPACES
MOVE TOTAL-OUTPUT-LINE-1 TO OUTPUT-RECORD
GO TO DISPLAY-PROGRAM-TOTALS-FINISH.
IF TOTALS-ARRAY-ID(DISPLAY-COUNTER) EQUALS SPACES
MOVE TOTAL-OUTPUT-LINE-2 TO OUTPUT-RECORD
GO TO DISPLAY-PROGRAM-TOTALS-FINISH.

DISPLAY-PROGRAM-TOTALS-FINISH.
WRITE OUTPUT-RECORD AFTER ADVANCING 1 LINE.
MOVE SPACES TO PROCESS-ID-1, CLIENTS-PROCESSED-1,
PROCESS-ID-2, CLIENTS-PROCESSED-2,
PROCESS-ID-3, CLIENTS-PROCESSED-3.
0405-DISPLAY-PGM-TOTALS-EXIT.
EXIT.

LEVEL 5

THIS ROUTINE IS CALLED FROM 0403 TO FIRST DETERMINE IF THE 551 FORM EXISTS FOR A CLIENT, AND IF SO,
LOAD IT INTO THE TEST ARRAY. ONCE A FORM IS FOUND, A FLAG IS SET INDICATING ITS PRESENCE FOR USE
IN A DEEPER LEVEL. A CALL IS MADE TO ANOTHER PARAGRAPH TO EXTRACT THE FORM DATA, PLACING IT IN WORKING
STORAGE. THE SUB-PARAGRAPH LOAD-551-DATA IS INVOKED SHOULD THE FORM EXIST. THIS ROUTINE WILL MOVE
THE APPROPRIATE DATA INTO THE TEST ARRAY IN ACCORDANCE WITH THE STATE TABLES OUTLINED IN THE SYSTEM
DOCUMENTATION MANUAL. THE FOLLOWING LIST DENOTES THOSE ROUTINES INVOKED FROM THIS PARAGRAPH:

* 0601-CHECK-FOR-ALL-FORMS
* 0602-READ-OUT-FORM
* 0701-READ-ERROR-DUMP (CONDITIONAL)

0501-SEARCH-LOAD-551.

MOVE 551 TO FORM-NAME-CHECK
PERFORM 0601-CHECK-FOR-ALL-FORMS THRU 0601-CHECK-FOR-ALL-FORMS-EXIT.
IF CLIENT-DATA-INDEX EQUALS ZERO
   MOVE 'N' TO 551-PRESENT
   GO TO 0501-SEARCH-LOAD-551-EXIT
ELSE
   MOVE 'Y' TO 551-PRESENT.
   OPEN INPUT FORM-ID-FILE.
   MOVE 551 TO FORM-ID-SEARCH-ITEM.
   READ FORM-ID-FILE INVALID
   CLOSE FORM-ID-FILE
   MOVE '0501-1' TO READ-ERROR-LCN
   MOVE FORM-ID-SEARCH-ITEM TO READ-ERROR-KEY
   MOVE 'FORM-ID NOT FOUND' TO READ-ERROR-DESC
   PERFORM 0701-READ-ERROR-DUMP THRU 0701-READ-ERROR-DUMP-EXIT
   STOP RUN.
   CLOSE FORM-ID-FILE.
   PERFORM 0602-READ-OUT-FORM THRU 0602-READ-OUT-FORM-EXIT.

LOAD-551-DATA.

MOVE FORM-DATA-RECORD-ITEM(START-PULL + 53) TO TEST-1-ITEM(1).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 60) TO TEST-1-ITEM(2).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 62) TO TEST-1-ITEM(3).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 63) TO TEST-1-ITEM(4).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 64) TO TEST-1-ITEM(5).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 43) TO TEST-1-ITEM(6).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 44) TO TEST-1-ITEM(7).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 45) TO TEST-1-ITEM(8).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 42) TO TEST-1-ITEM(9).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 29) TO TEST-1-ITEM(10).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 2) TO TEST-1-ITEM(11).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 3) TO TEST-1-ITEM(12).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 42) TO TEST-1-ITEM(13).

0501-SEARCH-LOAD-551-EXIT.

EXIT.

*******************************************************************************

*THIS ROUTINE IS CALLED FROM 0403 TO FIRST DETERMINE IF THE 552 FORM EXISTS FOR A CLIENT, AND IF SO,
*LOAD IT INTO THE TEST ARRAY. ONCE A FORM IS FOUND, A FLAG IS SET INDICATING ITS PRESENCE FOR USE
*IN A DEEPER LEVEL. A CALL IS MADE TO ANOTHER PARAGRAPH TO EXTRACT THE FORM DATA, PLACING IT IN WORKING
*STORAGE. THE SUB-PARAGRAPH LOAD-552-DATA IS INVOKED SHOULD THE FORM EXIST. THIS ROUTINE WILL MOVE
*THE APPROPRIATE DATA INTO THE TEST ARRAY IN ACCORDANCE WITH THE STATE TABLES OUTLINED IN THE SYSTEM
*DOCUMENTATION MANUAL. THE FOLLOWING LIST DENOTES THOSE ROUTINES INVOKED FROM THIS PARAGRAPH:
* 0601-CHECK-FOR-ALL-FORMS
* 0602-READ-OUT-FORM
* 0701-READ-ERROR-DUMP (CONDITIONAL)

0502-SEARCH-LOAD-552.

MOVE 552 TO FORM-NAME-CHECK
PERFORM 0601-CHECK-FOR-ALL-FORMS THRU 0601-CHECK-FOR-ALL-FORMS-EXIT.
IF CLIENT-DATA-INDEX EQUALS ZERO
   MOVE 'N' TO 552-PRESENT
ELSE
    MOVE 'Y' TO 552-PRESENT.
    OPEN INPUT FORM-ID-FILE.
    MOVE 552 TO FORM-ID-SEARCH-ITEM.
    READ FORM-ID-FILE INVALID
    CLOSE FORM-ID-FILE
    MOVE '0502-1' TO READ-ERROR-LOC
    MOVE FORM-ID-SEARCH-ITEM TO READ-ERROR-KEY
    MOVE 'FORM-ID NOT FOUND' TO READ-ERROR-DESC
    PERFORM 0701-READ-ERROR-DUMP THRU 0701-READ-ERROR-DUMP-EXIT
    STOP RUN.
    CLOSE FORM-ID-FILE.
    PERFORM 0602-READ-OUT-FORM THRU 0602-READ-OUT-FORM-EXIT.
    LOAD-552-DATA.
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 68) TO TEST-1-ITEM(14).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 68) TO TEST-1-ITEM(15).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 69) TO TEST-1-ITEM(16).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 70) TO TEST-1-ITEM(17).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 71) TO TEST-1-ITEM(18).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 72) TO TEST-1-ITEM(19).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 73) TO TEST-1-ITEM(20).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 74) TO TEST-1-ITEM(21).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 75) TO TEST-1-ITEM(22).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 76) TO TEST-1-ITEM(23).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 77) TO TEST-1-ITEM(24).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 78) TO TEST-1-ITEM(25).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 79) TO TEST-1-ITEM(26).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 80) TO TEST-1-ITEM(27).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 81) TO TEST-1-ITEM(28).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 82) TO TEST-1-ITEM(29).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 83) TO TEST-1-ITEM(30).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 84) TO TEST-1-ITEM(31).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 85) TO TEST-1-ITEM(32).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 86) TO TEST-1-ITEM(33).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 87) TO TEST-1-ITEM(34).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 88) TO TEST-1-ITEM(35).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 89) TO TEST-1-ITEM(36).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 90) TO TEST-1-ITEM(37).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 91) TO TEST-1-ITEM(38).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 92) TO TEST-1-ITEM(39).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 93) TO TEST-1-ITEM(40).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 94) TO TEST-1-ITEM(41).
    MOVE FORM-DATA-RECORD-ITEM(START-PULL + 95) TO TEST-1-ITEM(42).

0502-SEARCH-LOAD-552-EXIT.

EXIT.

******************************************************************************
* THIS ROUTINE IS CALLED FROM 0403 TO FIRST DETERMINE IF THE 651 FORM EXISTS FOR A CLIENT, AND IF SO,
* LOAD IT INTO THE TEST ARRAY.  ONCE A FORM IS FOUND, A FLAG IS SET INDICATING ITS PRESENCE FOR USE
* IN A DEEPER LEVEL.  A CALL IS MADE TO ANOTHER PARAGRAPH TO EXTRACT THE FORM DATA, PLACING IT IN WORKING
* STORAGE.  THE SUB-PARAGRAPH LOAD-651-DATA IS INVOKED SHOULD THE FORM EXIST.  THIS ROUTINE WILL MOVE
* THE APPROPRIATE DATA INTO THE TEST ARRAY IN ACCORDANCE WITH THE STATE TABLES OUTLINED IN THE SYSTEM
* DOCUMENTATION MANUAL.  THE FOLLOWING LIST DENOTES THOSE ROUTINES INVOKED FROM THIS PARAGRAPH:
* 0601-CHECK-FOR-ALL-FORMS
* 0602-READ-OUT-FORM
* 0701-READ-ERROR-DUMP (CONDITIONAL)

0503-SEARCH-LOAD-651.
    MOVE 651 TO FORM-NAME-CHECK
    PERFORM 0601-CHECK-FOR-ALL-FORMS THRU 0601-CHECK-FOR-ALL-FORMS-EXIT.
    IF CLIENT-DATA-INDEX EQUALS ZERO
        MOVE 'N' TO 651-PRESENT
        GO TO 0503-SEARCH-LOAD-651-EXIT
    ELSE
        MOVE 'Y' TO 651-PRESENT.
        OPEN INPUT FORM-ID-FILE.
MOVE 651 TO FORM-ID-SEARCH-ITEM.
READ FORM-ID-FILE INVALID
CLOSE FORM-ID-FILE
MOVE '0503-1' TO READ-ERROR-LCN
MOVE FORM-ID-SEARCH-ITEM TO READ-ERROR-KEY
MOVE 'FORM-ID NOT FOUND' TO READ-ERROR-DESC
PERFORM 0701-READ-ERROR-DUMP THRU 0701-READ-ERROR-DUMP-EXIT
STOP RUN.
CLOSE FORM-ID-FILE.
PERFORM 0602-READ-OUT-FORM THRU 0602-READ-OUT-FORM-EXIT.
LOAD-651-DATA.
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 25) TO TEST-1-ITEM(28).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 37) TO TEST-1-ITEM(29).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 42) TO TEST-1-ITEM(30).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 49) TO TEST-1-ITEM(31).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 50) TO TEST-1-ITEM(32).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 51) TO TEST-1-ITEM(33).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 52) TO TEST-1-ITEM(34).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 2) TO TEST-1-ITEM(35).
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 3) TO TEST-1-ITEM(36).
0503-SEARCH-LOAD-651-EXIT.

******************************************************************************
**THIS ROUTINE IS CALLED FROM 0403 TO FIRST DETERMINE IF THE 652 FORM EXISTS FOR A CLIENT, AND IF SO,**
**LOAD IT INTO THE TEST ARRAY. ONCE A FORM IS FOUND, A FLAG IS SET INDICATING ITS PRESENCE FOR USE**
**IN A DEEPER LEVEL OF A CALL IS MADE TO ANOTHER PARAGRAPH TO EXTRACT THE FORM DATA, PLACING IT IN WORKING**
**STORAGE. THE SUB-PARAGRAPH LOAD-652-DATA IS INVOKED SHOULD THE FORM EXIST. THIS ROUTINE WILL**
**THE APPROPRIATE DATA INTO THE TEST ARRAY IN ACCORDANCE WITH THE STATE TABLES OUTLINED IN THE SYSTEM**
**DOCUMENTATION MANUAL. THE FOLLOWING LIST DENOTES THOSE ROUTINES INVOKED FROM THIS PARAGRAPH:**
**
* 0601-CHECK-FOR-ALL-FORMS
* 0602-READ-OUT-FORM
* 0701-READ-ERROR-DUMP (CONDITIONAL)

0504-SEARCH-LOAD-652.
MOVE 652 TO FORM-NAME-CHECK
PERFORM 0601-CHECK-FOR-ALL-FORMS THRU 0601-CHECK-FOR-ALL-FORMS-EXIT.
IF CLIENT-NAME-INDEX EQUALS ZERO
MOVE 'N' TO 652-PRESENT
GO TO 0504-SEARCH-LOAD-652-EXIT
ELSE
MOVE 'Y' TO 652-PRESENT.
OPEN INPUT FORM-ID-FILE.
MOVE 652 TO FORM-ID-SEARCH-ITEM.
READ FORM-ID-FILE INVALID
MOVE '0504-1' TO READ-ERROR-LCN
MOVE FORM-ID-SEARCH-ITEM TO READ-ERROR-KEY
MOVE 'FORM-ID NOT FOUND' TO READ-ERROR-DESC
PERFORM 0701-READ-ERROR-DUMP THRU 0701-READ-ERROR-DUMP-EXIT
STOP RUN.
CLOSE FORM-ID-FILE.
PERFORM 0602-READ-OUT-FORM THRU 0602-READ-OUT-FORM-EXIT.
LOAD-652-DATA.
MOVE FORM-DATA-RECORD-ITEM(START-PULL + 44) TO TEST-1-ITEM(37).
0504-SEARCH-LOAD-552-EXIT.

*THIS PARAGRAPH IS CALLED FROM 0403 AND CONTAINS THE MAJOR WORK INVOLVED IN THIS LOGICAL LEVEL.
*THE ROUTINE INITIALIZES INDEX VARIABLES AND ENTERS INTO A SUB-LOOP TO BOUNCE THROUGH THE
*STATE TABLE. WHEN A CHARACTER IS READ, IT IS CHECKED TO DETERMINE IF IT CAN STAND ALONE, OR WHETHER
*IT IS PART OF A RESPONSE. ONCE THIS HAS BEEN VERIFIED, IT IS THEN CONVERTED TO A NUMERIC VALUE AND
*MAPPED DOWN TO AN INDEX IN THE STATE TABLE IN ACCORDANCE WITH COLUMN DESCRIPTORS FOUND IN THE SDN. AFTER
*THE STATE HAS BEEN PULLED FROM THE TABLE, THE VALUE IS FIRST CHECKED FOR END OF VALIDATION OR IT IS
*SENT TO ANOTHER PARAGRAPH FOR ERROR CHECKING. THE FOLLOWING LIST DENOTES THOSE ROUTINES INVOKED FROM
*THIS PARAGRAPH:

*  0603-CHECK-RESP-LENGTH
*  0604-CHECK-PROPERTY
*  0605-CHECK-PROPERTY-STATES
*  0606-CHECK-PROPERTY-STATES

0505-STATE-Lookup.

MOVE 0 TO FILE-INDEX.
MOVE 1 TO STATE.
STATE-Lookup-loop.
ADD 1 TO FILE-INDEX.
MOVE SPACES TO RESPONSE.
MOVE SPACES TO RESPONSE-Breakout.
IF FILE-INDEX > 43 GO TO 0505-STATE-Lookup-EXIT.
PERFORM 0603-CHECK-RESP-LENGTH THRU 0603-CHECK-RESP-LENGTH-EXIT.
PERFORM 0604-CHECK-PROPERTY-THRU 0604-CHECK-PROPERTY-EXIT.
PERFORM 0605-CHECK-PROPERTY-STATES THRU 0605-CHECK-PROPERTY-STATES-EXIT.
IF STATE = EXIT-STATE
GO TO 0505-STATE-Lookup-EXIT.
IF STATE = 0 DISPLAY
  'DATA ERROR. STATE= 0, INPUT= ', RESPONSE
GO TO 0505-STATE-Lookup-EXIT.
MOVE STATE TO LAST-STATE.
MOVE FILE-ARRAY-ITEM(LAST-STATE, COL) TO STATE.
MOVE 0 TO ERROR-STATE.
PERFORM 0605-CHECK-PROPERTY-STATES THRU 0605-CHECK-PROPERTY-STATES-EXIT.
IF ERROR-STATE = 1
  IF STATE = 0
    MOVE FILE-ARRAY-ITEM(LAST-STATE,28) TO STATE
  ELSE
    MOVE FILE-ARRAY-ITEM(LAST-STATE,28) TO STATE.
GO TO STATE-Lookup-loop.
0505-STATE-Lookup-EXIT.

**LEVEL 6**

**LEVEL 5**

**LEVEL 4**

**LEVEL 3**

**LEVEL 2**

**LEVEL 1**

**LEVEL 0**

*THIS ROUTINE CALLED FROM 0501. LOOPS THROUGH THE CLIENT-DATA-FILE SEARCHING FOR A POINTER TO THE
*FORM BEING EXAMINED. A VALUE OF 0 IS RETURNED IF THE CLIENT DOES NOT HAVE THE FORM.

0601-CHECK-FOR-ALL-FORMS.

MOVE 1 TO CLIENT-DATA-INDEX.

CHECK-FOR-FORM.

IF CLIENT-DATA-INDEX > 3
MOVE ZEROS TO CLIENT-DATA-INDEX
GO TO 0601-CHECK-FOR-ALL-FORMS-EXIT.

IF CLIENT-DATA-FORM(CLIENT-DATA-INDEX) EQUALS FORM-NAME-CHECK
GO TO 0601-CHECK-FOR-ALL-FORMS-EXIT.

ADD 1 TO CLIENT-DATA-INDEX.
GO TO CHECK-FOR-FORM.

0601-CHECK-FOR-ALL-FORMS-EXIT.

EXIT.

******************************************************************************

*THIS PARAGRAPH, CALLED FROM 0501 THRU 0504, IS USED TO ENTER THE APPROPRIATE FORM DATA FILE AND
*COMPUTE THE BEGINNING AND ENDING LOCATIONS OF THE FORM. THE INTERNAL PARAGRAPH COMPUTE-PULLS IS A
*STANDARD ROUTINE IN THE ADES SYSTEM, USED IN CONJUNCTION WITH THE FORM-ID-FILE WHICH
*DETAILS EACH FORM. THE FOLLOWING LIST DENOTES THOSE ROUTINES INVOKED FROM THIS PARAGRAPH:
*
* 0701-READ-ERROR-DUMP (CONDITIONAL)

0602-READ-OUT-FORM.

MOVE FORM-ID-DATA-NAME TO FORM-DATA-NAME-FILE-NAME.
OPEN INPUT FORM-DATA-FILE.
PERFORM COMPUTE-PULLS.
READ FORM-DATA-FILE INVALID.
CLOSE FORM-DATA-FILE.
MOVE '0602-' TO READ-ERROR-LCN.
MOVE FORM-DATA-FILE-ACTUAL-KEY TO READ-ERROR-KEY.
MOVE 'NO FORM FOUND' TO READ-ERROR-DESC.
PERFORM 0701-READ-ERROR-DUMP THRU 0701-READ-ERROR-DUMP-EXIT.
STOP RUN.

CLOSE FORM-DATA-FILE.
GO TO 0602-READ-OUT-FORM-EXIT.

COMPUTE-PULLS.

DIVIDE CLIENT-DATA-LOC(CLIENT-DATA-INDEX) BY FORM-ID-RECORDS-BLOCK
GIVING FORM-DATA-FILE-POSITION-KEY.

IF FORM-DATA-FILE-POSITION-KEY EQUALS ZERO
MOVE FORM-ID-RECORDS-BLOCK TO FORM-DATA-FILE-POSITION-KEY.
ELSE
ADD 1 TO FORM-DATA-FILE-POSITION-KEY.

COMPUTE START-PULL EQUALS FORM-ID-CHAR-RECORD * 
(FORM-DATA-FILE-POSITION-KEY - 1) + 1.

COMPUTE END-PULL EQUALS FORM-ID-CHAR-RECORD * 
FORM-DATA-FILE-POSITION-KEY.

0602-READ-OUT-FORM-EXIT.

EXIT.

******************************************************************************

*THIS ROUTINE IS CALLED FROM 0505 TO ASCERTAIN THE LENGTH OF RESPONSE FOR PARTICULAR QUESTIONS. THE
*FILE-INDEX VARIABLE IS USED AS THE INDICATOR. SHOULD THIS INDEX MEET ONE OF THE CRITERIA IN THIS
*Routine, THE NEXT N CHARACTERS ARE MOVED INTO THE VARIABLE USED FOR STATE TABLE LOOKUP. IF ALL CONDITIONS
*FAIL, THE RESPONSE IN ONE CHARACTER ONLY, AND IS SIMPLY MOVED TO THE VARIABLE.

0603-CHECK-RESP-LENGTH.
IF FILE-INDEX = 3
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-1
  ADD 1 TO FILE-INDEX
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-2
  ADD 1 TO FILE-INDEX
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-3
  GO TO 0603-CHECK-RESP-LENGTH-EXIT.
IF FILE-INDEX = 7
  MOVE 0 TO RESPONSE-1
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-2
  ADD 1 TO FILE-INDEX
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-3
  GO TO 0603-CHECK-RESP-LENGTH-EXIT.
IF FILE-INDEX = 11
  MOVE 0 TO RESPONSE-1
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-2
  ADD 1 TO FILE-INDEX
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-3
  GO TO 0603-CHECK-RESP-LENGTH-EXIT.
IF FILE-INDEX = 16
  MOVE 0 TO RESPONSE-1
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-2
  ADD 1 TO FILE-INDEX
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-3
  MOVE RESPONSE TO RESPONSE-2N
  GO TO 0603-CHECK-RESP-LENGTH-EXIT.
IF FILE-INDEX = 18
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-1
  ADD 1 TO FILE-INDEX
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-2
  ADD 1 TO FILE-INDEX
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-3
  MOVE RESPONSE TO RESPONSE-3N
  GO TO 0603-CHECK-RESP-LENGTH-EXIT.
IF FILE-INDEX = 32
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-1
  ADD 1 TO FILE-INDEX
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-2
  ADD 1 TO FILE-INDEX
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-3
  GO TO 0603-CHECK-RESP-LENGTH-EXIT.
IF FILE-INDEX = 35
  MOVE 0 TO RESPONSE-1
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-2
  ADD 1 TO FILE-INDEX
  MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-3
  GO TO 0603-CHECK-RESP-LENGTH-EXIT.
MOVE TEST-1-ITEM(FILE-INDEX) TO RESPONSE-1.
0603-CHECK-RESP-LENGTH-EXIT.
EXIT.

************************************************************
*THIS ROUTINE, INVOKED FROM OS05, WILL TRANSLATE THE INPUT DATA INTO A COLUMN VALUE FOR TABLE LOOKUP.*
*FOR TABLE DEFINITION, PLEASE CONSULT THE SDN.

0604-CONVERT-DATA.

IF RESPONSE IS ALPHABETIC
  MOVE RESPONSE-1 TO CHAR-FIELD
  COMPUTE COL = NUM-FIELD - 32
  GO TO CONVERT-DATA-FINISH.
 IF (RESPONSE IS NUMERIC) AND
   (RESPONSE > 26)
   MOVE 27 TO COL
   GO TO CONVERT-DATA-FINISH.
 IF (RESPONSE IS NUMERIC) AND
   (RESPONSE = 0)
   MOVE 29 TO COL
   GO TO CONVERT-DATA-FINISH.
 IF (RESPONSE IS NUMERIC) AND (RESPONSE NOT > 26)
   MOVE RESPONSE TO COL
   GO TO CONVERT-DATA-FINISH.

CONVERT-DATA-FINISH.

IF (RESPONSE EQUALS SPACES) OR (RESPONSE = '0 ')
  MOVE 28 TO COL.

0604-CONVERT-DATA-EXIT.

EXIT.

*THIS PARAGRAPH IS CALLED FROM 0505 TO ASCERTAIN IF THE CURRENT STATE REQUIRES SPECIAL EVALUATION.

*THE FOUR STATES INVOLVED ARE 40, 50, 80, AND B2. THIS ROUTINE WAS WRITTEN DUE TO THE COMPLEXITY OF THE
*STATE TABLE TO CIRCUMVENT THE NECESSITY OF THIS ROUTINE.

0605-CHECK-SPECIAL-STATES.

IF (STATE = 40) AND (552-PRESENT EQUALS 'Y')
   AND (RESPONSE NOT EQUAL SPACES) AND (RESPONSE NOT EQUAL '0 ')
   COMPUTE TEST-VAR = RESPONSE-2N + RESPONSE-3N
   IF TEST-VAR > 180
   MOVE 14 TO COL
   ELSE
   MOVE 25 TO COL.

IF (STATE = 50) AND (552-PRESENT EQUALS 'Y')
   AND (TEST-1-ITEM(23) = TEST-1-ITEM(24))
   AND (RESPONSE NOT EQUAL SPACES) AND (RESPONSE NOT EQUAL '0 ')
   MOVE 25 TO COL.

IF (STATE = 80) AND (552-PRESENT EQUALS 'Y') AND (552-PRESENT EQUALS 'Y')
   AND (RESPONSE NOT EQUAL SPACES) AND (RESPONSE NOT EQUAL '0 ')
   IF TEST-1-ITEM(14) = TEST-1-ITEM(37)
   MOVE 25 TO COL
   ELSE
   MOVE 14 TO COL.

IF (STATE = 92) AND (RESPONSE NOT EQUAL SPACES)
   AND (RESPONSE NOT EQUAL SPACES) AND (RESPONSE NOT EQUAL '0 ')
   IF (TEST-1-ITEM(24) EQUALS SPACES)
   OR
   (TEST-1-ITEM(24) = TEST-1-ITEM(43))
   OR
   (TEST-1-ITEM(43) = SPACES)
   MOVE 25 TO COL

178
ELSE

MOVE 14 TO COL.

0605-CHECK-SPECIAL-STATES-EXIT.

EXIT.

******************************************************************************
*THIS ROUTINE, INVOKED FROM 0605, WILL SIMPLY CHECK THE CURRENT STATE TO DETERMINE IF AN ERROR HAS BEEN
*DETECTED. THE SIMPLISTIC STRUCTURE ALLOWS EASY ANALYSIS AND MAINTENANCE. SHOULD AN ERROR STATE
*BE FOUND, THE RESPECTIVE ERROR MESSAGES ARE LOADED INTO THE MEMO RECORD AND WRITTEN TO THE OUTPUT FILE.

0606-CHECK-ERROR-STATES:

IF (STATE = 6)

MOVE '551' TO MEMO-FORM
MOVE '26' TO MEMO-QTN1
MOVE '28' TO MEMO-QTN2
MOVE 'EMPLOYMENT STATUS' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 7)

MOVE '551' TO MEMO-FORM
MOVE '26' TO MEMO-QTN1
MOVE '30' TO MEMO-QTN2
MOVE 'EMPLOYMENT STATUS' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 11)

MOVE '551' TO MEMO-FORM
MOVE '18B' TO MEMO-QTN1
MOVE '19' TO MEMO-QTN2
MOVE 'RE-ADMISSION STATUS' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 23)

MOVE '551' TO MEMO-FORM
MOVE '15' TO MEMO-QTN1
MOVE '18A' TO MEMO-QTN2
MOVE 'REASON NOT FOR SERVICE DESC' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 24)

MOVE '551' TO MEMO-FORM
MOVE '11' TO MEMO-QTN1
MOVE '15' TO MEMO-QTN2
MOVE 'DESCRIPTION NOT FOR CATEGORY' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 33)

MOVE '551-552' TO MEMO-FORM
MOVE '18A' TO MEMO-QTN1
MOVE '341(1)' TO MEMO-QTN2
MOVE 'DRUG NOT FOR REASON' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 37)

MOVE '552' TO MEMO-FORM
MOVE '341(1)' TO MEMO-QTN1
MOVE '341(1)' TO MEMO-QTN2
MOVE 'PRI DRUG/LAST 30 DAYS' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 39)
MOVE '552' TO MEMO-FORM
MOVE '341(2)' TO MEMO-QTN1
MOVE '341(5)' TO MEMO-QTN2
MOVE 'IN EXCESS OF 180 DAYS' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 46)
MOVE '552' TO MEMO-FORM
MOVE '341(1)' TO MEMO-QTN1
MOVE '341(4)' TO MEMO-QTN2
MOVE 'DRUG SHOULD NOT BE PRESCRIBED' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 49)
MOVE '552' TO MEMO-FORM
MOVE '341(1)' TO MEMO-QTN1
MOVE '342(1)' TO MEMO-QTN2
MOVE 'PRIMARY/SECONDARY RELATIONSHIP' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 54)
MOVE '552' TO MEMO-FORM
MOVE '342(1)' TO MEMO-QTN1
MOVE '342(4)' TO MEMO-QTN2
MOVE 'DRUG SHOULD NOT BE PRESCRIBED' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 57)
MOVE '552' TO MEMO-FORM
MOVE '341(1)' TO MEMO-QTN1
MOVE '35' TO MEMO-QTN2
MOVE 'OTHERS PROBLEM' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 63)
MOVE '651' TO MEMO-FORM
MOVE '16' TO MEMO-QTN1
MOVE '18' TO MEMO-QTN2
MOVE 'REASON DOES NOT MATCH TYPE' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 67)
MOVE '651' TO MEMO-FORM
MOVE '22' TO MEMO-QTN1
MOVE '24' TO MEMO-QTN2
MOVE 'EMPLOYMENT STATUS' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 70)
MOVE '651' TO MEMO-FORM
MOVE '22' TO MEMO-QTN1
MOVE '25' TO MEMO-QTN2
MOVE 'EMPLOYMENT STATUS' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 74)
MOVE '651-652' TO MEMO-FORM
MOVE 'DRUG' TO MEMO-QTN1
MOVE '301(1)' TO MEMO-QTN2
MOVE 'SERVICE IS NOT FOR DRUG' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 79)
MOVE '552' TO MEMO-FORM
MOVE '301(1)' TO MEMO-QTN1
MOVE '301(3)' TO MEMO-QTN2
MOVE 'DRUG SHOULD NOT BE PRESCRIBED' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 81)
MOVE '552-552' TO MEMO-FORM
MOVE '341(1)' TO MEMO-QTN1
MOVE '301(1)' TO MEMO-QTN2
MOVE 'PRIMARY DRUGS NOT EQUAL' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 83)
MOVE '552-552' TO MEMO-FORM
MOVE '342(1)' TO MEMO-QTN1
MOVE '302(1)' TO MEMO-QTN2
MOVE 'SECONDARY DRUGS NOT EQUAL' TO MEMO-CLASS
GO TO DISPLAY-ERROR-MSG.

IF (STATE = 0)
MOVE 1 TO ERROR-STATE
GO TO 0606-CHECK-ERROR-STATES-EXIT.

DISPLAY-ERROR-MSG.
MOVE 1 TO CLIENT-MSG.
MOVE 1 TO ERROR-STATE.
ADD 1 TO ERROR-COUNT.
MOVE MEMO-LINE TO OUTPUT-RECORD.
WRITE OUTPUT-RECORD.
MOVE SPACES TO BASIC-CLIENT-INFO.
ADD 1 TO PROGRAM-ERRORS.
GO TO 0606-CHECK-ERROR-STATES-EXIT.
EXIT.

**********************************************************************************************************
LEVEL 7
**********************************************************************************************************

* THIS ROUTINE, CALLED FROM 0602 IS USED TO DISPLAY THE KEY FOR THE FORM-DATA-FILE AND ALL PERTINENT
  CLIENT DATA WHEN AN INVALID READ IS DONE ON THE FORM-DATA-FILE. THE SYSTEM IS EXITED HERE
  FOR DATA ANALYSIS AND SYSTEM RESTART. THE FOLLOWING LIST DENOTES THOSE ROUTINES CALLED FROM THIS
  PARAGRAPH:
*  MACRO BELL (WMU BUSINESS COLLEGE)
  0701-READ-ERROR-DUMP.
  ENTER MACRO BELL USING 20.
  DISPLAY ".
  DISPLAY 'READ ERROR LOCATION: 'READ-ERROR-LCN.
  DISPLAY 'KEY: 'READ-ERROR-KEY.
  DISPLAY 'DESCRIPTION OF ERROR: 'READ-ERROR-DESC.
  0701-READ-ERROR-DUMP-EXIT.
  EXIT.
BIBLIOGRAPHY


