A FRAME BASED ON-LINE REFERENCE PACKAGE

By

TERRY JAY JOHNSON

Bachelor of Arts
University of Michigan
Ann Arbor, Michigan
1970

Master of Library Science
Louisiana State University
Baton Rouge, Louisiana
1979

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1985

A FRAME BASED ON-LINE REFERENCE PACKAGE

Thesis Approved:

M J Folk
Thesis Adviser

Donald D Fisher

Kelvin L. Davis

Norman N Durham
Dean of the Graduate College

PREFACE

This paper is a discription of a frame based on-line reference package applied to computer program debugging. It includes an authoring program, descriptions of the program modules, and user's guides to both the reference package and the authoring program.

I would like to express sincere gratitude to my major adviser, Dr. Michael J. Folk, for his guidance, motivation, and invaluable help. I am also thankful to Dr. Donald D. Fisher and Dr. Sharilyn A. Thoreson for their insightful suggestions and encouragement during the course of this work. An extra thank you must go to Dr. Kelvin L. Davis for agreeing to serve on my committee as a last minute substitute.

My deepest gratitude to my family, Cora, John, and Kimberley for their encouragement, for their love, and for just being there.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Development of the Concept

One of the most valuable utilities most operating systems provide is an on-line command reference manual. When the UNIX user invokes this by the command, man, followed by the name of the command requested, the system will display a brief description of the requested command's format and function. UNIX also provides a computer assisted instruction package(CAI) invoked by the command, learn, that is supposed to aid the user in understanding system commands and protocols. Considered as reference tools, each of these has weaknesses and strengths, and to some degree, the strengths and weaknesses are complementary. The strengths of one may compensate for the weaknesses of the other. This project intends to present a model of a reference tool that will combine aspects of computer assisted instruction with those of on-line reference manuals to produce a product that has many of the advantages of the two methods and hopefully few of the disadvantages. A C language debugging package will be developed to demonstrate this tool in practical application.

Before examining this reference tool in greater detail, the strengths and weaknesses of the individual components will be examined more closely starting with computer assisted instruction henceforth referred to as CAI. As the name implies, computers are used as either a supplement or as an alternative to conventional classroom instruction. To a degree, it may be unfair to note the weaknesses of this method as a reference tool since its purpose is not reference but instruction. However, it is the view of the author that a good reference tool should involve more teaching than most existing tools do now, and most instructional tools should also be more useful as references than they are now. With this as background, CAI will be examined as a teaching tool.

Since classroom instruction usually involves lecture, CAI has most often been compared with lecture (Steinberg, 1984). Its advantages in this respect have been noted as:

1) CAI can require a response from every user instead of the usual articulate few who make most classroom responses.

2) The answers and mistakes are private--no one need fear public embarrassment.

3) The presentation speed is controlled by the student and can thus be individually paced and not set by the instructor or the rest of the class.

4) Statistics of individual performance can be monitored by the instructor who then has an idea of the progress of the class.

5) The lessons are available when the equipment is available, not just when the instructor and classroom are.

Disadvantages of CAI when compared to classroom lecture:

1) CAI implementation usually have limited power to answer questions. CAI will either not answer questions or do so in the most rudimentary fashion. This is probably due to the complexity of natural language processing for interpretation of questions. Typically, such courses refer the student to a human instructor whenever they feel the need for extra help.

2) No group discussion is possible. This can be a valuable teaching aid. A peer can often express things in a manner more comprehendable than an instructor simply because the levels of experience are closer. Group discussion may suggest alternative methods of viewing topics that might not occur to the student or an instructor, be that instructor man or machine.

3) Normally, only visual presentation of material is used. While this is not intrinsic to the CAI method, audio presentations are rare.

4) There are no visual cues possible between student and instructor. In the classroom, these can provide valuable feedback to the instructor in pacing the presentation and discovering what may need further explanation.

Although not often mentioned, CAI can be compared with another visual student paced medium, the textbook. In some sense, it seems strange that the textbook is so seldom men-

tioned because it and CAI share so many characteristics. In another sense, it may not seem unusual at all since CAI as usually implemented is more similar to an on line lecture interspersed with exercises than it is to an on line textbook.

It is now time to examine in more detail the advantages and disadvantages of currently practiced CAI compared with the textbook.

CAI Advantages Over the Textbook:

1) CAI can force a response.

2) The computer can provide feedback and reinforcement that a textbook can not.

3) CAI is present whenever the computer is operating, and so is available when a text may not be.

4) Depending upon the implementation, CAI has the potential of easier modification to add or delete text.

CAI Disadvantages Compared to a Textbook:

1) Presentation is generally sequential. There is no opportunity to skip around and look ahead.

2) Presentation is unidirectional. Usually there is no way to review a previous section under user control. Learn, for instance, enables a student to indicate a previous checkpoint and repeat from there, but fine control is not possible.

3) Usually, nothing equivalent to a table of contents or index is available to allow direct access only to a particular topic.

4) While a textbook can be used for future reference after course work is completed, CAI implementations are specialized to one purpose: teaching.

5) A student may copy especially relevant sections of a text for future reference, whereas one may only take notes in a CAI session. This is another similarity between CAI and lecture.

6) A book may be browsed, but CAI, by forcing responses, compels a particular behavior pattern. Questions must be answered to progress through the course; the user is unable to take a relaxed view of the material, pausing where his interests or needs require.

At this point, a picture of this "ideal" reference tool is beginning to form. Clearly, CAI has a role in providing explanation in depth. Most of CAI 's deficiencies with regard to lecture can be resolved by providing some means of communication between users. This reference tool, therefore, should provide an integral means of communication. This tool should also be very book-like in that the control of direction and manner of use of the reference tool should be under the user's control. To guide him in the exercise of this control, he should be provided with an index. The user now decides what will be studied and to what degree; he has the capability to browse and skip around in many direc-

tions. However, by giving the user this power, the implementation strays far from traditional CAI. Responses are no longer required before progressing and so it is not possible to monitor a student's progress. Since monitoring and package control of user actions have been noted as advantages of CAI, perhaps an explanation is required as to why they have been dropped from the reference tool. The answer is simple: those are instructional advantages. In a reference environment, they conflict with the freedom desired in this tool. Since the author believes that exercises are valuable for the practice and reinforcement of concepts, they will be included, but performance of them is optional.

On-line reference aids such as man or help are not intended to instruct; they exist for reference only with the implicit assumption that the user is somewhat familiar with the material being presented. Therefore, text is usually condensed and difficult to comprehend. Additionally, examples are seldom present and the format relatively rigid.

Occasionally, a user may try to access a topic that is a subtopic under another heading. If the user is unaware of this, he is puzzled and frustrated. He has no index or list of topics available on-line with which to search for related topics.

The modifications to CAI that been have mentioned previously provide a tool that already addresses the flaws just noted. What is not yet available, though, is a tool that provides the normal function of the on-line reference manu-

al: providing a brief summary of function and format. Yet this is all that some users require. Meeting this demand implies that the tool should be multi-level with one level equivalent to the on-line manual and another level similar to a CAI presentation.

Now that requirements for this tool have been defined, the implementation method can be made more concrete. The text is stored in individual files; each of which contain, at most, the amount of text that may be displayed upon a terminal screen. A terminal screen contains twenty-four lines although only about twenty lines are available for text with the other four being used for menu and prompt display. Each file of twenty lines must connected with similar files so that a presentation sequence may be followed. This linking to other files is accomplished by includeing within each file references to other connecting files. These connection references are known as links and the whole structure of linked files is a linked list. Because of the multilevel aspect of this tool, the files or nodes of the major linked list may, in turn, head other linked lists. This overall structure can be summarized by saying that the text is stored in a linked list of lists. In the terminology used henceforth in this paper, one of these nodes or files will be referred to as a "frame" and will contain linking information as well as the text that would be representable upon a terminal screen.

Presentation of the data is accomplished by a driver program that operates by traversing this framework of linked frames. As the driver program traverses, it presents the textual information of each frame upon the terminal screen. The path of frame traversal is determined by the user entered commands given at each frame.

There are several types of frames. The types of frames used for storing text depend upon the type of text they store. These types are as follows:

1) Summary frames provide a brief description of a topic with the textual style and purpose similar to that of an on-line manual.

2) Explantion frames contain in depth explanations of the topics presented in the summary frame, and can be accessed by traversing from a summary frame. The text is of the level of CAI, providing explanation to someone who is relatively unfamiliar with the topic.

3) Example frames contain examples taken from the context of actual applications and can be reached from explanation frames. Examples are not included within the instruction frame in order to keep frame sizes of approximately screen size and not to interfere with the flow of concentration developed within the instruction frame.

4) Exercise frames contain simple examples intended to reinforce concepts mentioned in the instruction frame. Like the example frame, the exercise frame is considered subsidiary to the explanation frame and is reached from the expla-

nation frame.

5) Communication frames allow on-line interaction between the user and the people who control the package's operation and contents.

6) Index frames contain an alphapbetical listing of the topics in the package and allow the user direct access to the frame of his choice.

In addressing the previously noted requirements, the following features represent the approach taken in this package:

1) Multilevel presentations: The user may stay at the summary level, browsing or reviewing, or he may go to the instruction level on a particular topic. Exploration of example and exercise frames is optional. The user may go to whatever level suffices for his needs.

2) Modular design: By using linked list design, nodes may be inserted or modified with little difficulty.

3) Indexing: The user may access any topic directly through the index without having to proceed sequentially through extraneous information.

4) Examples: Examples illustrate implementations of the topic under discussion. Viewing of examples is optional, allowing the hurried user to bypass any of them that may be irrelevant to his needs.

5) Exercises: This is a feature whose performance is optional but whose presence allows reinforcement of concepts through practice.

6) Copying: Each frame will allow the option of copying the frame into the student's directory.

7) Multidirectional traversal: The user may traverse the frames forward, backward and in some cases laterally. Review is possible with the user controlling the extent.

8) Communication: A bulletin board frame allows communication between users and the tool designer and between instructors and students. Complaints, examples, suggestions, and information may all be passed along. Feedback, communication, and group discussion are now possible without having to learn the mechanisms of phone, msg, or mail.

9) Bookmark: By invoking the bookmark feature, a user can save the location of the particular frame of interest and can return there without having to remember a location identifier to be invoked at a future time.

10) Application independent: The driver may be applied to any set of properly constructed files. This allows this reference package to be used for any subject.

Application

The example application chosen for the on-line reference tool is that of a debugging reference package. This application can illustrate the tool and at the same time provide a much needed utility. Bugs are errors in computer programs that interfere with the proper execution of the program. The process of detecting and correcting bugs is known as debugging. Lukey (1982) views the debugging pro-

cess as a problem solving process where the user attempts to define a hypothesis about the error from clues given. The user tries to find the bug that supports his hypothesis. The more clues a user has, the more effectively he can debug. One major purpose of the the application is to provide as many clues as possible.

This application will provide the following benefits for students and instructors:

1) Error interpretation: The C compiler diagnostic messages and the run time messages can be cryptic and misleading.

2) Instructor aid: Instructors and student assistants who are seeing the same type errors in a programming assignment may indicate the problem and solution in the bulletin board. This may allow office hours to be used for other concerns.

3) Student scheduling: The student no longer has to match his schedule with that of the instructor. The bulletin board is not time dependent and communication may flow in both directions.

4) Interstudent communication: Class "experts" need not be hounded whenever they appear in the terminal room and may attend to their own assignments instead of solving everyone else's.

5) System communication: The student does not have to recall all the various communication protocols for system messages.

6) Availability: Expert help is on line and available to all.

Major Modules for the Debugging Application:

1) Aid for the interpretation of common C compiler error messages.

2) Introduction to lint.

3) Introduction to the run time debugging tool, adb

4) Lore - knowledge gained from past experience

5) Bulletin board

a) General

b) Class

1) Instructor's notes

2) Student to instructor messages

CHAPTER II

REVIEW OF LITERATURE

The subject matter of this paper covers several areas of study. The following areas are most relevant to this study: CAI, frame oriented systems, debugging in general, active computer aided debugging, system supplied aids. CAI is a common acronym for "computer assisted instruction" or using the computer to assist in instruction. This acronym will be used in the rest to this paper when referring to this topic. The method of presentation used for this project most closely resembles the frame oriented interface structures under study in some quarters which is discussed more fully later. Since the application of this system is debugging, a survey was made of past studies in this area. More directly relevant, though, are the attempts to use the computer as an active aid in debugging. The computer goes beyond the traditional role of giving compiler and run time error messages and actually tries to interpret the errors. System supplied aids include the various debuggers and other aids such as lint (in UNIX). The rest of this survey will examine each of these topics in more detail.

Computer Assisted Instruction

Since CAI is one of the major influences upon this project, research findings from this field can be a valuable design aid. One cornerstone of CAI research concerns feedback. Education research has shown that the feedback to the user after a response is a key factor in the efficacy of retention. What form should this feedback take? Current research(Hartley and Lovell, 1984) has overturned some traditional theories in the field. At one time, it was felt that the mere acknowledgement that an answer was correct answer was sufficient feedback if the material was presented in small steps. More recent research indicates that this approach is inadequate. A more effective approach is to use information as feedback. Information about answers is more effective in increasing performance than feedback that only indicates that an answer was correct.

Another relevant question is: who controls the learning environment -- the student or the computer? The weight of evidence indicates that users feel better when they control but that they do not make effective decisions regarding their abilities and needs.

Gaines(1984) recommends that CAI systems be constructed so that users may learn about a system by using it. Interestingly enough, and in contradiction to the previous paragraph, he considers that at the present state of the art, the user should dominate the computer. In accordance with good programming practice, he recommends that all

presentations and response requirements be as uniform as possible. Also, the user should be able to query in depth.

Simpson(1984) recommends that programs present menus of command choices in order to lessen the demands upon the user's memory. In addition, he feels that menus help the user orient himself with regard to the system. The major drawback is that of the extra resources required for the storage and presentation of these menus. This sort of overhead is also the reason that Friend et al.(1984) recommend that graphics be used very sparingly.

Frame Oriented Systems

Frame oriented systems rely upon the presentation of pages of text or in the terminology of the field, "frames" to the user for operation. The prime example of this is ZOG (McCormick and Alkscyn, 1984)(Alkscyn and McCormick, 1984) which is a general purpose shell whose operation depends upon the use of menus. ZOG is considered frame based which means that the user is presented information a page at a time and each page is considered a single unit. Nothing is to be scrolled and every attempt to keep units of information limited to a single page. ZOG was originally developed to be an interface between several different computer systems. The user, instead of having to learn the unique characteristics of several different operating systems with the inevitable resulting confusion, only had to learn one which was kept deliberately simple. One application of the

ZOG approach has been that of a distributed database spread across several different machines.

To the user, working with ZOG appears to be traversing a set of frames. Each frame includes a menu with the set of operations that can be performed and also the possible frames that can reached from the present one. This can promote exploration and browsing. Akscyn and McCormick compare conventional database usage to fishing with a pole and line and ZOG to swimming around among the fish. The user goes to the data and not vice versa. The use of paging versus scrolling has received other support in the literature. Barry et al. (1982) and Schwarz et al. (1983) found that users much preferred paging or windowing to scrolling.

Debugging

The cost of programming errors or "bugs" is high. It has been estimated that "debugging" a program takes three times longer than writing it (Gould, 1975). Despite this, the level of research is still somewhat rudimentary. The problem appears to be that no one can quite get a good grasp of a worthwhile approach. It is a problem that all programmers wish would get solved, but no one is certain how. Typically, bugs are classified into two broad categories, syntactic and non syntactic. Syntactical errors are usually defined as those detected by a compiler. After Boies and Gould (1974) studied syntactical errors, the subject has not received a great deal of attention. Boies and Gould did a

statistical analysis of programs submitted for compilation at the Thomas Watson Research Center to study the frequency of syntax errors. The results indicated that only about one sixth of the programs contained syntax errors on first compilation. Boies and Gould felt that this result was probably typical and since then, this type of error has received relatively little attention. It has been felt that efforts would be more productive if directed elsewhere. Miller and Thomas (1977) in reviewing studies of syntax errors concluded that developing more comprehensive syntax checkers may not be cost effective since many checking facilities already available are unused. One has to wonder how much of this lack of use is related to ignorance, to complicated checking aids, or to ineffective tools.

Gannon(1978) classified errors somewhat differently with more emphasis upon the types of errors that occur. Using a subset of ALGOL (ST), Gannon found the following types of errors most frequent:

1) Declaring variables in one procedure and requesting them in another.

2) Redeclaring a global variable.

3) Using a global variable when a local variable was wanted.

4) Misspelling variable names.

5) Not initializing variables.

6) Exceeding array limits.

7) Incorrectly performing a case statement.

8) Passing parameters in incorrect order.

9) Passing the wrong number of parameters.

10) Becoming confused with embedded structures.

11) Not initializing control variables in an iterative structure.

12) Not modifying control variables in an iterative structure.

13) Mismatching parentheses.

14) Not matching variable usage to declaration.

Clearly, these are common errors, but again there has been little work replicating or expanding upon these, so one can make few judgements about how typical these are in other situations.

Brooke (1982) extends the nonsyntactic error classification into two parts:

1) Incorrect formulation of algorithms so that they will never work.

2) Inadequate formulation of algorithms so that they will work within certain limits but fail when these limits are exceeded.

Brooke does not give any information about how this new formulation may be applied nor what it may gain, but it does indicate a possible new direction in debugging research.

Experimental studies of program debugging have been around for quite a while, but consistent interpretations and applications are not so common. Gould and Drongowski (1974)

have made one of the more complete studies of the subject. Gould and Drongowski gave thirty experienced programmers twelve one page FORTRAN listings, each with one of three types of bugs (array bugs, bugs in loops, and bugs in assignment statements). In addition, the programmers were divided into five groups based upon the amount of extra information they were given along with the listings. One group, to be used as a baseline, was given no information, another group was given the I/O for their listings, another group was given the I/O plus the I/O that should be produced if the program ran correctly; another group was told the class of error that was present, and the last group was given the line number of the error.

The subjects were to find the bug and identify its nature. The results showed that experience helped tremendously. Subjects found errors up to three times faster when given a different bug in a program that they had previously debugged. It was also found that assignment bugs were the most difficult to detect. The experimenters felt that detection of an assignment bug required a more thorough knowledge of a program than the other two and thus was more difficult under these conditions.

The effects of the different sorts of aids upon debugging time was unexpected. The time required to find bugs for the two groups with I/O hints was greater than the time for the group with no aid at all. This result was interpreted as demonstrating the adaptability of programmers to

various conditions. However, the no aid group detected erroneous bugs sixty percent more often than the other groups. The final two groups were able to use their hints to selectively examine their listings. Not surprisingly, the group given the line number was about twice as fast as the other groups. Given the magnitude of the hint, the authors speculated as to whether a twofold increase represents some sort of upper limit on possible speedup.

Gould (1975) replicated the experiment apparently only giving I/O to the group and provided interactive aids. His results were similar to the results previously obtained. One surprise was that even when interactive aids were available for use in the experiment, they were not used. Unfortunately, the nature of these interactive aids was never specified so it is difficult to attempt explanation of this fact. Gould believes that the debugging process follows a particular pattern. First, programmers use whatever clues they have available to develop a hypothesis and then try to verify their hypothesis from the program. Wanting to find the bug with the least effort, programmers tend to ease into their programs trying to find the bug at the highest levels and in the easier portions of the program. It is only later, and with reluctance, that they will study the program in depth for understanding.

In his study of the debugging process, Weiser (1982) developed the concept of program "slices". By using clues to generate hypotheses, programmers subsequently attend

only to those portions of their programs relevant to their hypothesis. They "slice" away the non-essentials. He believes that it is possible to tailor debugging aids to follow these concepts better than has been done previously.

Computer Aided Debugging

Using the computer to aid in debugging is not new. Traditionally, there have been interactive debugging tools to step through programs. These applications are relatively passive and will be discussed in the following section. The present section concerns the computer as a more active participant in the debugging process. Many self-study on-line tutorials of languages require the student to write code and then let the tutorial evaluate the result in some fashion. One sophisticated example of such a system is the Stanford BIP (BASIC Instruction Project) (Barr, Beard, and Atkinson, 1976), intended to teach the BASIC language. Since the program is intended to be used without the presence and aid of an instructor, thorough error detecting capabilities were required. Error detecting capabilities cover "syntax and execution time errors, program structure errors detectable before execution but involving more than the syntax of one line." Additionally, they have "added clarifying messages for each error, including examples of correct and incorrect statements, which the student receives upon request." This can easily be seen to be a built-in version of what is presently being proposed in this project as an add-on aid.

It must be noted that BIP is atypical in its sophistication. Most such systems give very little aid with messages being terse or nonexistent.

Another approach that has attracted attention is the use of some form of artificial intelligence. The GPSI (Laursen, 1981) project developed at the University of Illinois at Urbana uses an expert system to aid students in the debugging of FORTRAN program syntax errors. Expert systems attempt to emulate experts by applying "rules" to detect pattern matches between a request and information held in a rule base. Succinctly, one might think of an expert system as an intelligent database (Stefik et al. 1982). It is "intelligent" in the sense that it must in some fashion manipulate its request parameters in order to apply them to its rule base to generate a response to the user.

A typical GPSI session requires that the student bring a listing of the program with the error messages so that GPSI may be used to help interpret these messages. The error message consists of an error number with a short message (usually vague in meaning) and a pointer to a section of the line that supposedly caused the error. GPSI presents the user with a list of error numbers with messages and a list of pointer positions. The user types in the combination of these that his program exhibits for a particular line. GPSI then interprets this pattern and attempts to generate a diagnostic message appropriate for this error pattern. The user is presented with with this message and prompted to

agree or disagree with its relevance. If the user agrees, the same process is repeated with the next error. If not, GPSI attempts to generate another possible message if it has one in its repertoire and the process begins anew. If there are no more messages, GPSI repeats what it has already presented until the user gives up querying for more information. Such a system has certain prerequisites that limit its application. The compiler must not produce error messages derived from previous errors; otherwise the patterns become much more difficult to detect. A related requirement is that messages be accurate and consistent, again to facilitate consistent pattern generation. All too often, this is not the case.

The error diagnostic capabilities of GPSI and BIP have limitations. GPSI has only twenty-two rules and BIP's repertoire focuses upon the errors typically generated by its lessons. The answer in each case is simply to build up the capabilities and here lies a major weakness. Modifying BIP means modifying the compiler which is usually no small task. Changing GPSI requires that more rules be added. Laursen confesses " A major problem with GPSI is that it is very tedious to generate the rules..." One advantage of GPSI is that the human being is involved at least to the extent of judging the relevance of responses and possibly triggering other responses. This is not true of BIP although the depth of explanation may provide more clues to the user in understanding possible causes of error.

An approach that has elements of both of the above is the Bug Finder (Bonar et al., 1982) which is part of the Meno II language tutoring system. Used alone, it assists in batch grading of programming assignments or it can directly aid individuals with error detection. Like BIP, it works directly upon programs but it does so in a manner somewhat like GPSI, attempting to derive patterns that trigger further action. The process consists of four steps. First, the program is parsed to generate an abstract representation. In the second step, the abstract representation is interpreted as much as possible to try to determine what is supposed to happen and this purpose is annotated to the relevant section. An example annotation is "running total." In the third step, the information thus far generated is compared with "plans" or patterns stored in the Bug Finder to determine how well the program does what is intended. For instance, the "running total" program section might be compared with a "running total " plan that checks for initialization, misapplication of running variables, "off by one errors", and so forth. Finally, discrepancies between student programs and the plans are interpreted as bugs and appropriate messages are generated. Again, the power of this scheme depends upon the library of plans and bugs available. The generality of this kind of scheme depends upon how easily new plans can be added.

System Supplied Aids

The term "system supplied aids" refers to the compiler diagnostic messages, debuggers, and similar aids. Most systems contain such aids. This discussion will concentrate upon UNIX supplied utilities. UNIX supplies some help to the user; the most basic aid, of course, is the C compiler with its error messages. Of more interest is lint (Johnson, 1982) which checks for good programming practice and warns of potential trouble areas. For run time error checking, two tools are available from UNIX; adb and in more recent versions sdb. Both are of the class known as debuggers which means that users may set breakpoints in a program such that, in execution, the program stops at these breakpoints and reports the values of variables. They have other capabilities as well(Maranzano and Bourne, 1982), but, as with other members of the class, they are considered "arcane, complicated, and indispensable," (Kernighan and Pike, 1984). Because of complaints such as these, other debuggers have been developed that are much easier to use. Unfortunately, they are not readily available. Steffens (1984) discusses a C debugging aid called CTRACE which allows an easy trace of variables through a C program. Unfortunately, it is an in-house debugger at Bell Labs and not yet available to other users. It should be noted that, by tracing only particular variables, this type of aid most resembles that recommended by Weiser in his previously mentioned concept of program slicing. This is the same sort of concept advocated by El-

liot (1982) applied to other languages. Cargill (1984) mentions a sophisticated debugging aid available on the Blit terminal (Last price quoted was five thousand dollars each for this kind of terminal.)

The final conclusion implied by the previous discussion of debugging tools is that the debugging application of the reference tool developed for this project has a definite function that is not adequately addressed elsewhere. Existing systems that might fulfill this purpose are difficult and cumbersome to change. Other more conventional tools are either difficult to understand, limited in scope, or not readily available. The rest of this paper develops a method that attempts to use a frame based textual presentation, influenced by CAI, to develop an easily modifiable system applied to debugging with potential application in many other areas.

CHAPTER III

METHODS AND PROCEDURES

Design Criteria

The design criteria for this frame based on-line reference package were that it be user controlled, simple to use, and easily modified. The attempt was to provide a tool, not a master. A tool is an instrument that the user controls; something that controls the user is a master. Since this reference tool is a frame based system that operates through frame traversal, the application of the principle of user control implies that the pattern of frame traversal is under user control. This implementation follows this principle faithfully by providing the user with a menu in each frame and visiting the frame indicated by the command chosen.

The second design criteria was ease of usage. A tool is not effective if no one feels comfortable using it. This package was made to be used with a minimum of study and direction; another reason that the frame based method of presentation was used. As used in the ZOG system and in this system, command menus are presented at the foot of each frame in order to eliminate recall and format problems. The presentations and responses were made as uniform as possible. Almost all of the commands require only the entrance

of a single letter or number.

Another important requirement was simplicity of modification. As noted in the previous chapter, more sophisticated systems exist that perform the same task as the application of the package does, but all are difficult and tedious to modify. This system is quite flexible with regard to modification. The methodology of doing so will be explored in the authoring section of this chapter.

Implementation

Environment

The on-line reference package was written in the C language under the UNIX operating system that runs on a Perkin-Elmer 3230 Computer. As implemented, the UNIX environment is more that just a background; it is an integral part of the package. Several of the programs interface with system utility commands to accomplish their purposes. This approach has the benefit of using previously written tools rather than starting from nothing. Since this implementation is file based and many utilities are designed to operate upon files, many of the system file commands can be directly applied.

Concurrency

Using system tools does have some risk. By using tools that are not developed by the package author, one has the risk of unanticipated and possibly damaging side effects.

One such side effect is concurrent file access. The UNIX system does not normally provide exclusive file access. This means that more than one person may be reading from or writing to the same file simultaneously. Since reading does not modify the file, more than one user may read without danger. Writing is another situation altogether since there is a real danger of blending the outputs from several sources into the same output file. For this reason, write access is severely limited in this implementation.

Writing is allowed under two circumstances: writing done by a privileged user or writing done by a system utility. The privileged users are those with access rights to the source directory of the package. The tasks performed by these people include writing text (discussed in the authoring section), writing to the bulletin board (discussed in the communications section), or modifying the underlying program. Since only a limited number of people will have directory access privileges, it is hoped and expected that only one of them would be performing these tasks at once. Practically, a particular person would be delegated to perform these tasks. Thus, there would be little or no risk of concurrent writing to a particular file.

The second writing situation uses system utilities to allow the ordinary user to send messages to predesignated people. The message facility of the package works as front end to the UNIX system utility, mhmail, which in turn acts as a front end to the sending utility, post. What these

utilities send are completed packets of information. The system utility handles all scheduling and writing. The utility avoids the blending of packages by sending serially the completed packets.

There is also the problem of concurrently running programs. What happens if more than one person is using this package at the same time? Again, UNIX provides non-exclusive execution. "Procedural code in all programs produced by the C compiler is reentrant and sharable."(Deitel, 1984). The user gets an image to use as if it were in the user's directory. Since, as noted previously, writing is handled separately, all other file handling in the program involves reading, and reading presents no conflicts even with multiple users.

Menu Strategy

The menu strategy follows the ZOG approach for the use of menus. This means that each frame presents the user with all the allowable choices permitted from that frame. The major benefit is that users do not have to recall what to do next; all the choices are before them. Presentation of all options means that new users or those who have been away from the system do not need to constantly refer to a manual or scribbled notes from a previous session. Users are able to use all the capabilities of the system, not merely those that are more easily remembered through frequency of use.

The use of menus has several disadvantages. Making available all options causes an appreciable amount of screen space to be used for the menu. To minimize screen space usage and also present the options requires a horizontal presentation. While the nature of the situation compels this approach, it may not be the ideal method. Heines (1984) sets these requirements for the use of horizontal layouts:

> 1) The menu is not the major screen feature.
> 2) The overall screen image is to be preserved.
> 3) There are a small number of menu options.
> 4) Each menu option is limited to one or two words (p. 68).

```
cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
please enter next command:
```

Figure 1. Typical Menu Display

As can be seen from the example in Figure 1, Heines' item three is violated. The result is not aesthetically pleasing, somewhat cluttered, and difficult to read. In such cases, vertical layouts would be preferable; they are more pleasing to the eye, easier to read, and possibly less confusing. Although vertical menus would be preferable, use of them in the present situation would restrict the screen available for text far too much and cause text to be spread between frames even more than is presently the case.

On many terminals it is possible to highlight text by such means as underlining or reverse video. Where this ca-

pability has been defined for the terminal in use, it has been used throughout the frames to highlight keywords. Highlighting has been used to differentiate the menus from the rest of the text and partially mitigate the disadvantages of using horizontal menus. The command descriptions are highlighted and in capital letters. The commands are not highlighted and are lower case letters.

Reference Package Structure

The basic unit for the reference tool package is the frame. As has noted previously, a frame is the text that fits in the space provided by a terminal screen. In operation, this system presents the user with a series of frames, the order and timing of presentation being under the user's control. Each frame has a similar configuration. The upper portion of the frame is text pertaining to a topic. The lower portion of the frame presents a menu with the possible command options for this frame. These commands allow the user to perform certain functions or to travel to other frames. A more detailed examination of these commands will be presented in the frame command section later.

Frame Storage

Many alternatives exist for the storage of the frames in the computer. One method that was considered (and rejected) was to store the text in one large file from which the driver program could have extracted the appropriate sections for each frame. Indeed, this approach is used with the package's index processing modules. The index is configured as one large file from which the index programs extract the relevant portions for presentation as frames. However, the index has certain special features that make it more amenable to this approach. The index is serial and the material is extracted in a forward or backward manner from contiguous portions of the file. There is no need for the elaborate record keeping that might be required for other types of application that require jumping about the file.

The text material on the other hand may not be seen in the order in which it has been stored. From any frame, a user has a choice of any of several new frames. While this requirement makes it more difficult to store the material in one large file, it does not make it impossible. The basic problem with this approach is that of complexity which violates the design criteria of simplicity of modification. This approach to storage requires the use of sophisticated record keeping to keep track of offsets into the file for particular frames. Insertion, deletion, and modifications can change the length of a particular sections and change the offsets to the beginnings of particular frames. To ad-

just for all these changes can become difficult: there are simply too many pieces of information to readily account for.

The approach used by the package stores each frame's text in a separate file. Since each file requires some overhead for the system to maintain its location and maintain various statistics, this approach may not be considered economical in terms of system resources. On the other hand, the system handles all retrieval and storage problems. Conceptually, this method of storage matches the frame concept more closely. Modification becomes much less complex since all frames are independent units. Insertion becomes a simple matter of creating a new file and creating the appropriate links in the other files involved in this frame path. Deletion becomes the simple matter of removing a file and resetting links. The concept of the overall structure being a list of lists becomes easier to grasp and manipulate when all the nodes of the list are files and all links merely the naming of appropriate files. Frame modification becomes a simple matter of text editing; maintenance is straightforward. Overall, the disadvantage of extra resource requirements for the multifile text storage arrangement is more than offset by the advantage of simpler maintenance.

Frame Set

Seen from the point of view of the package structure, the basic unit is an entity known as a frame set which are the frames devoted to a particular topic. A full or normal frame set consists of four major frame types: a summary frame, an explanation frame set, an example frame set, and an exercise frame. The word, set, is used in the previous sentence to denote the possibility of multiple frames of these types in the frame set. Recall that a frame is limited to the amount of information that may be presented upon one display screen. To explain or give examples about a topic may require more information than can be displayed upon a screen at one time. Conversely, the amount of material that is available upon a topic may not require the full frame set. In this case the "abbreviated" frame set may be used. Abbreviated means that that all four frame types are not present in a particular frame set. For instance, an explanation frame is not required if the topic can be covered adequately in the summary frame. Exercises may not be appropriate if the topic is completely self evident. Individual variations depend upon individual situations. However, a summary frame must be present to provide continuity and connections to the rest of the framework.

A full frame set has a particular structure that ought to be made more clear before proceeding any further. The structure of this reference tool, with the exception of certain special frames that will be mentioned later, is that of

a series of frame sets connected with each other through the summary frames (Figure 2). The summary frame then forms an "outer layer" with the other frame types forming "inner layers". In a full frame set, the summary frame connects to the first explanation frame of the explanation frame set. Example frames connect to the explanation frame(s). Also connected to the last explanation frame is the exercise frame. A pictorial representation of this is present in Figure 3. An example of an abbreviated frame is present in Figure 4.

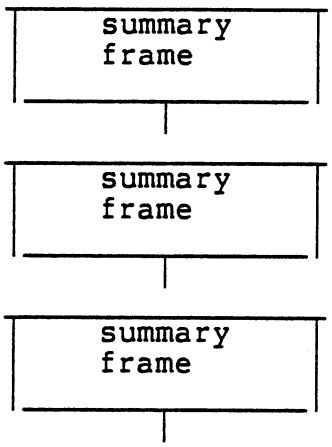

Figure 2. Connecting Framework of Reference Package

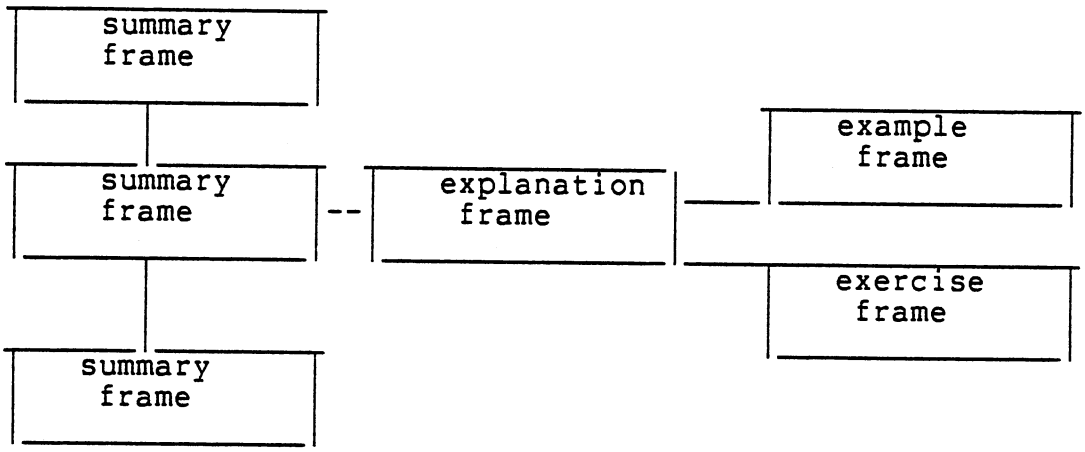

Figure 3. Full Frame Set

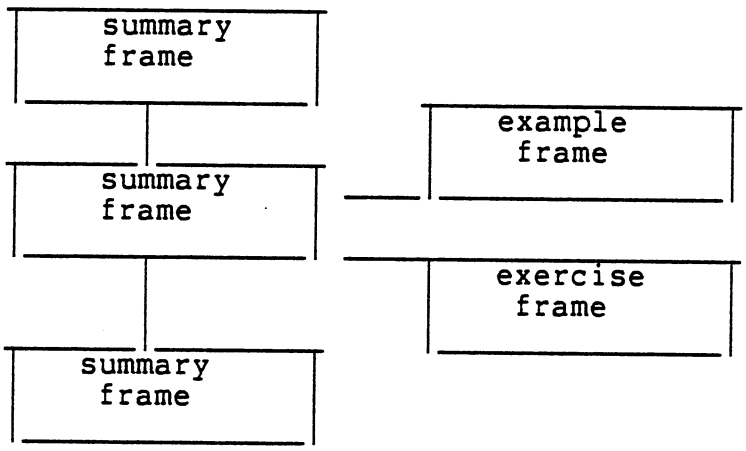

Figure 4. Abbreviated Frame Set

Frame Types

Summary Frame: The user is presented with a summary of the topic for this frame set. The intention is to limit

this summary to one frame. Greater detail is presented in the explanation frame. The summary frame performs several functions in this reference tool. The summary frame is a very important component of the reference package and has the following functions:

1) Synopsis: Some users require only a summary of the topic. Through previous experience with this tool or through knowledge of the subject matter, a brief reminder is all that is necessary. Further information is not needed.

2) Retention: Studies of learning retention (Mayer, 1981) show that subjects presented with some sort of conceptual framework for new material had greater subsequent recall of material. Indeed, it has been theorized that part of the learning process is the construction of such structures. Thus, the summary initially presented in this frame is also intended to provide the beginning of a knowledge structure to aid in the retention of the material.

3) Connection: Summary frames connect frame sets.

4) Foundation: Summary frames are the base from which the rest of the frame set grows.

Explanation Frame: These frames explain the topic in greater depth than the summary frame. The intent of this frame type is to instruct those who may have little background in the subject. Since the explanation may be lengthy, a chain of explanation frames may be required.

Example Frame: These are intended to be used as an adjunct to the explanation frame. They illustrate the points

alluded to in the explanation frames. Again, there may be numerous examples relating to a particular point so there may also be a set of these frames. Since there may be different points relating to the topic, there may be more than one set of examples deriving from the explanation set.

Exercise Frame: Passive reading of a topic is not enough for many users. An application of a concept usually aids substantially in it retention. Concept application also tests whether the concept has been learned. Exercises are provided as means of applying concepts just learned to aid in retention and test the amount of learning that has taken place.

Index Frame: This is a special frame type that does not belong to the frame set structure. This frame presents the topics of summary and explanation frame sets sorted alphabetically and numbered. The user discovers the name of the frame, enters the associated frame number and the package invokes the indicated frame.

Error Frame: This frame is not part of the frame set structure either. By mistake or mischief, some users will use inappropriate commands when prompted. This will cause a visit to the default or error frame. The user is then presented an error message and given instructions about exiting the default frame.

Program Description

Frame Based Reference Package

Data Structure

If the files containing the frame materials are considered as nodes with information about the nodes preceding and succeeding, it can be seen that each frame is a member of a linked list. Conceptually, the the linked list of summary frames defines the framework. Each summary frame or node can in turn head a list of explanation nodes each of which, in turn, can head lists of examples and exercises. Overall, the structure can be considered a linked list of linked lists. All linked lists except the linked list of summary frames are circular; that is, the node following the last node is the beginning node for the list. And since there are links in both directions, it is a doubly linked list. Thus, in computer science terminology, the entire structure can be considered close to a doubly circular linked list of linked lists, although links to other members of the frame set prevent this from being a pure form.

Operation

Operating the reference package involves the traversal of nodes using various sources of information to determine the next node. Each information frame begins with a table containing the names of predecessor and successor nodes as well as the names of other specified nodes in the frame set.

Since the user may leave the current frame set and go to other destinations and then want to return, the names of the present and previously visited nodes are also retained in the global file. The global file also maintains a stack of the names of marked frames that support the mark command (defined in the next section). From any of these sources, the name of the next frame to be presented may be obtained. In addition, there is one more source, the index file. The index file keeps the operation of the package from becoming purely sequential in nature. By invoking the index routine, the user is able to get the names of summary and explanation files and directly access them without traversing an intermediate path.

A broad description of package commands follows. A more detailed description of using the package is given in Appendix C.

Frame Instruction Set

The set of frame instructions or commands can most effectively be examined if they are divided into categories. One convenient categorization labels commands as being either universal or context dependent. Universal commands are those available from any of the basic frame set type frames. Context dependent commands are those which are legal for execution only in particular situations or "contexts." Some of context dependent commands are valid only in full frame sets and not in some abbreviated sets. The detailed examination

of the commands given below should make the distinction between the two types very clear.

The universal commands may be further subdivided into those that perform a function or task and those that cause a new frame to be visited. All of the context dependent commands involve visiting a different frame.

Universal Task Commands. These commands, executable from all frames, do not involve changing frames.

h - Gives a brief explanation of each package command. This is the help command for the package.

l - Shows the frame path from the beginning frame to the user's present frame. The intention is that the user will be able to orient himself.

t - Shows the table of contents. A list of all frame set topics is displayed. The topic of the user's current frame set will be highlighted if the terminal has these capabilities. Again the user is able to orient himself in a different and possibly more useful manner than that afforded by the "l" command.

m - Invokes the message sending routine that sends a message to the instructor or other designated person.

c - Copies the text of the present frame into the user's directory under the name frame.copy.

k - Marks the present frame to enable future direct return to the marked frame. A user may have up to nine marked frames at any one time. These frames are revisited in reverse order of marking; that is, the last frame marked is

the first revisited.

Universal Frame Switching Commands. These commands are executable from all frames and result in changing frames.

q - Causes on-line reference package to stop execution.

b - Visits the frame on the list that precedes the present frame.

g - Returns to the marked frame on top of the marked stack.

p - Revisits the most recently visited frame. The distinction between this command and "b" defined previously may a bit subtle at first. The command "b" visits the predecessor of the frame as defined by the way the list is constructed. If a frame has been accessed in some manner other tan traversing the list in a forward direction, "p" will not access the same frame as "b." Additionally, with this instruction the user may easily move from the error frame.

i - Visits the index node. The index frame enables the user to discover the name of and directly visit particular topic frames.

Frame Dependent Commands. These commands depend upon the situation and may not be present in all frames. Their legality for a particular frame is shown by their presence in the menu. The distinction is simple: if the command is present in the menu, it is legal from that frame.

e - Visits the explanation frame of a frame set. This command is not allowed from an explanation frame nor is it

permitted from an abbreviated frame set which does not include an explanation frame.

s - Visits the summary frame. This is not allowed from a summary frame.

x - Visits the exercise frame. Except in an abbreviated frame set, this is possible only from the explanation frame. Many abbreviated frame sets visit this frame directly from the summary frame.

<number> - Visits the first member of the indicated example set. Typically, whenever a section of a topic is covered, there may be an example set for that topic. Within the set of explanation frames for the frame set, there may be several topics, each with its own set of example frames. The on-line reference system presently allows nine example sets for any frame set. This means nine sets of examples not merely nine examples. These frames are accessible from the explanation frames or summary frames in the case of an abbreviated frame set. The user accesses an example set by entering the appropriate number for the particular set desired. The text mentions these numbers in its presentation of the topic.

Frame Communications Facilities

As noted in the first chapter, communication facilities are very desirable in a tool such as this one. The ability to query an instructor about a topic, the ability to complain, and the ability to make constructive suggestions

depend upon communications. The UNIX operating system has efficient communication methods available which include several message sending utilities: msg, send, mail, and write . All have two major drawbacks. First, a user must know how to use them, and more importantly, they are not available within the reference package. To use them, the user must leave the package even though the topic may be intimately concerned with what is happening within the package.

To get around these difficulties, a facility is available within the package invoked by the command "m". Once invoked, the user is put into a frame with two choices, to read a bulletin board or to transmit a message. The read component is quite simple. Once invoked, the bulletin board is presented. Writing to the bulletin board is restricted to those who have directory privileges in the package's home directory in order to avoid concurrency problems, but reading is open to all. The bulletin board provides a quick method for the instructor of a course to communicate with all the students with reference to a particular problem related to the reference package's topics.

The danger of concurrent writing during the use of the transmit facility has been avoided by using the system utility mhmail. The transmit facility acts as a front end setting up the parameters for mhmail. The destinations are preset and limited to the instructor, teaching assistant, and research package designer. The user is prompted with

these and chooses one. Names can be added or changed as desired since the values are in global storage.

The next prompt gives instructions for entering a message. After the message has been entered, the user must decide whether he wishes to append a file to this message. The user has the capability to to send files or programs that illustrate particular points pertinent to the topics of this package. Since the present application of this tool involves debugging of programs, the file sending facility would appear to be potentially very useful in this context. Any named files are appended to the previous message which may well be explanatory in nature.

Once this is completed, the information is incorporated into the mhmail command and sent. The user is returned to the main program. It should be noted that since the receiver has bulletin board writing privileges, there is the possibility of developing a moderated on-line discussion among the user population.

Frame Authoring System

Someone must write the materials that make up the frames for this package. Each frame has two sections that must be completed. The first section contains the information about the "links" to other frames. The second section contains the text for the frame; the material that appears on the screen to the user.

There are certain protocols to be followed for each section. A tool called "author" handles many of these details. (Henceforth, in order to avoid confusion, the word "author" refers to the program that helps build the frame. The human who is building the frame will be referenced as "the writer.") Author prompts the writer for all necessary information; all the writer must do is respond. The initial prompts concern the frame type and name. From the answers received, author constructs a frame name derived from the user given name and a particular suffix which depends upon the frame type.

Suffixes are important to an index making program which includes only those files with suffixes indicating that they are explanation or summary files. These files are processed to extract the subject, name, and type of frame from this file and enter this information into the index. The suffixes of the example and exercise files exclude them from being processed. Since example and exercise frames branch from particular summary or explanation frames, exclusion of them from the index results in very little loss of practical information. However, their inclusion in the index has the potential of making the index larger, more unwieldy, and ultimately less useful.

As noted in the initial paragraph of this section, the beginning portion of the frame defines the frame's position with regard to other frames. In other words, this section contains the links to other frames. During execution of the

reference tool, this information is extracted and stored in a global table accessible to other programs of the package. Each frame, therefore, must contain all necessary values. Again, author handles the details querying the user for any information that is not readily available from other sources. The writer responds either with the relevant information or the default response. The default response signals that there is no relevant value for this link, and author responds by filling in the name of the error frame.

Once the preliminary section of setting links is finished, the text can be entered. The writer has two options in doing so: text may be entered at the terminal through author, or an existing file may be appended by author. The option chosen depends upon the situation. Very likely, example or exercise material will be appended, and explanation and summary material will be written in. Regardless of entry method, because of screen limitations, the material should not exceed twenty lines. As an aid in keeping text within the line limit, author provides line number prompts for those entering text from the terminal. No such aids are possible for appended file, so the user must edit these files accordingly.

A more detailed explanation of authoring with operating instructions is present in Appendix D, "AUTHORING GUIDE FOR THE ON-LINE REFERENCE PACKAGE."

Debugging Application: Methodology of Error Selection

The demonstration application of this program has been that of a debugging tool. The strategy has been to provide additional information concerning selected compiler error messages that can serve as extra clues to aid the user in debugging. Part of the problem in building this application is that of selection. What errors should be chosen and what information should be given about them?

The obvious approach is to find those errors which are the most common and whose messages are the least informative. Finding the most common errors may not be an easy task. The types of programming problems and assignments have a great deal to do with the types of errors seen. Obviously, students working on problems involving the use of data structures will see a different spectrum of errors than a class doing input and output problems. Because of the difference in error producing situations, representative errors may be difficult to define.

For the present study, error messages were generated by two approaches. The first approach used relatively simple programs that were initially correct, and errors of different types were systematically introduced into them. It was felt that most errors in actual practice were of the simple mistyping type. These include misspelling and omission of punctuation in vital areas. Some support for the frequent occurrence of these types of errors comes from Pierson and Horn(1984) where the majority of their reported

COBOL errors were of this type. Compiler error messages were produced using a black box approach. Errors were introduced into the programs, and the programs were fed into the compiler to see what error messages would be produced. The resulting error messages were then analyzed for underlying principles from which it would be possible to write explanations for the reference package.

Laursen(1982) used a similar approach to generate errors in the GPSI project. He notes two major weaknesses in this method. First, there is no way to obtain all the errors in an error class. Secondly, it is possible that errors generated in this fashion may be highly unlikely to occur in actual situations. But in fact, GPSI does relatively well handling the errors brought to it despite being based on error messages generated in this fashion. Similarly, the present project seems to have the more common messages represented in the output received.

Recognizing that the previous method was limited by the imagination of the error generator, another more realistic method was used to generate other error messages. The author collected errors made in his own programming assignments and solicited from others some of their unusual errors. Although not exhaustive, this method seemed to add significantly to the variety of errors in the model.

Two other methods for suggesting errors are worthy of note. One method is to look at the results of researchers. Pierson and Horn(1984) have been mentioned; another study is

that of Gannon (1978). Some of the errors that he mentioned have already been generated, and others are not detectable by the C compiler. Some of the remaining errors remain to be tested and provide the basis for further research.

The other method is to obtain an accurate measure of possible errors and their frequency by collecting and statistically analyzing incorrect programs. This remains a fruitful area for future research.

As a substitute for the statistical approach, it is hoped that the users will take advantage of the built-in communication facilities of the package to send information to the system maintainers. Feedback from users could enable the system to grow and change, reflecting their new information. Since the system is so simple to modify, new information could easily be incorporated into it and eventually it might account for a high proportion of the error situations presented to it.

CHAPTER IV

SUMMARY AND FUTURE WORK

On-Line Reference Package Synopsis

On-line references often are difficult for the novice to comprehend. Many features could be added to make the concepts more understandable. As it is, even experienced users are somewhat confused and often read only to get a starting idea. From there, they experiment until they finally comprehend what was intended initially. CAI, with its rich background, can aid in making reference entries more understandable as much through its philosophy as techniques. However, CAI suffers from shortcomings as well, especially with regard to communication with instructors and the designing of courses. The present on-line package solves the shortcomings of both by using a multilevel presentation with communication facilities built in. The sophisticated user can see summaries and not be overwhelmed with extraneous detail. The more inexperienced user can receive deeper explanation with exercises and examples. Rather than forcing the user into the lock step method of instruction, the user is given, as with a book, freedom to roam and browse. In addition, the user is given direct access capabilities of the traditional on-line reference system. A communication

system was added to allow communication between the user population and those who design and maintain the package and its text.

The application was that of an on-line debugging aid for debugging student C language programs. Numerous examples were added to illustrate the topics.

Limitations of On-Line Reference Package

Some of the limitations of the application of this study arise from the specificity of its implementation. Although it uses standard C commands, the UNIX operating system is an integral part of the package. Use of system features is a two edged sword: it has the benefit of using existing, proven tools, but the disadvantage of requiring a particular operating system. This package must be run on systems that use a UNIX operating system.

A related limitation is that the application deals with the error messages of a particular compiler on a particular machine. Obviously, to transfer this package to another machine with another compiler will involve some adaptation. The relevance and wording of particular messages may differ from computer to computer.

Another limitation is the standard one that applies to developing any learning or reference package: the development process is very labor intensive for an expert. The writing of text, examples, and exercises can take an appreciable amount of effort.

The package has not yet been applied to a user population. The limitations and deficiencies that practical usage inevitably reveal about any computer program will remain undiscovered until that time.

Future Project Considerations

The present application is that of a debugging aid. There are two separate elements here, the implementation and the application. Each has implications for further work.

The complete application as envisioned in the first chapter remains incomplete. Each module could be expanded. At present the module concerning compiler error messages is the most complete. The section on the C debugging aid, lint, contains primarily introductory material. The modules on the C debuggers and "lore" are minimal. To make this tool complete for its intended purpose, these sections should be expanded in the future.

There remains the problem of whether there is some size of subject matter beyond which this application should not venture. The limiting factor is the size of the screen and the amount of material that can be presented at one time. The limit is approximately twenty lines of text more or less. A large amount of material would take an appreciable amount of time to work through although mitigating this aspect considerably is the fact that as a reference text, it is unlikely that a user would go through it from front to back. This brings a related problem to the fore: the more

topics there are, the larger the index will be. At twenty lines per screen, there is probably some limit as to the number of topics that ought to be dealt with in one package. There is some point, possibly around ten or more pages of an index that might justify a split of material or an automated index.

The implementation is suitable for uses other than the present application. For instance, many of the less obvious system utility commands could be cast in this format with gain in clarity. Clearly, many of these would benefit from more explanation and more examples.

The menu has possibilities for improvement. For instance, if all available terminals had the capabilities of reverse video, the letter of the command could be highlighted and all extraneous spaces and dashes could be removed. There is the possibility of allowing the removal of the menu altogether for advanced users who may find it a hindrance rather than a help.

The authoring system could be modified to be more automated and limit the amount of knowledge that the writer must be familiar with. At present, many of the repetitive and obvious prompts have been eliminated with the authoring system itself deriving the requisite information. Clearly, more progress can be made in this regard to require even less knowledge about the package from the writer.

Portability

With a view toward portability, most constants and implementation features have been put into the global file "frames.h". Other applications would require that this be modified for their own usage. Certainly, such things as the start frame, full-path names, and the name of the index will change in another application; most of the other globals would probably remain constant.

SELECTED BIBLIOGRAPHY

Akscyn, R. M. and McCracken, D. L. "The ZOG Approach to Database Management." Pittsburgh, PA: Carnegie-Mellon University, 1984.

Barr, A., Beard, M., and Atkinson, R.C. "The Computer as a Tutorial Laboratory: the Stanford BIP Project." International Journal of Man-Machine Studies 8, 5 (1976), 567-596.

Boies, S. J. and Gould, J. D. "Syntactic Errors in Computer Programming." Human Factors 16, 3 (1974), 253-257.

Bonar, J., Ehrlich, K., Soloway, E., and Rubin, E. "Collecting and Analyzing On-line Protocols From Novice Programmers." Computer Research Methods and Instrumentation 14, 2 (1982), 203-209.

Brooke, J. B. "Tools for Debugging Computer Programs - How Much Do They Help?" in J. Rasmussen and W. B. Rouse (eds.), Human Detection and Diagnosis of System Failures. New York and London: Plenum Press, 1982, 87-109.

Bury, K., Boyle, J., Every, R., and Neal, A. "Windowing versus Scrolling on a Visual Display Terminal." Human Factors 24, 4 (1982), 385-394.

Cargill, J. A. "Debugging C Programs With the Blit." AT&T Bell Laboratories Technical Journal 63, 8 (1984), 1633-1645.

Deitel, H. An Introduction to Operating Systems. Reading,Massachusetts: Addison-Wesley Publishing Company, 1984.

Elliot, B., "A High-level Debugger for PL/I, Fortran, and Basic." Software-Practice and Experience 12, 4(1982), 331-340.

Friend, J. and Milojkovic, J. "Designing Interactions Between Student and Computer." in Walker, D. F. and Hess R. Instructional Software : Principles and Perspectives for Design and Use. Belmont, CA: Wordsworth Publishing Co.,1984, 143-150.

Gaines, B. "Technology of Interaction - Dialogue of Programming Rules." in Walker, D. F. and Hess R. Instructional Software : Principles and Perspectives for Design and Use. Belmont, CA: Wadwworth Publishing Co. ,1984 115-121.

Gannon, J. D. "Characteristic Errors in Programming Languages." Proceedings of the 1978 Annual Conference of the ACM. Washington D.C. : ACM, 1978, 570-575.

Gould, J. D. "Some Psychological Evidence on How People Debug Computer Programs." International Journal of Man-Machine Studies 7,1, (1975), 151-182.

Gould, J. D., and Drongowski, P. "An Exploratory Study of Computer Program Debugging." Human Factors 16, 3 (1974), 258-277.

Hartley, J. R. and Lovell, R. "The Psychological Principles Underlying the Design of Computer Based Instruction Systems. " Instructional Software Principles and Perspectives for Design and Use. Belmont, CA: Wadsworth Publishing Co.,1984, 38-56.

Heines, J. M. Screen Design Strategies for Computer-Assisted Instruction. Bedford,MA: Digital Press, 1984.

Johnson, S. C. "Lint, A C Program Checker." in Wollongong Group Edition VII Workbench : Programmer's Manual Volume 2. Oceanport, NJ: Perkin-Elmer Corporation, 1982, 1-11.

Kernighan, B. W. and Lesk M. E. "Learn - Computer-Aided Instruction on UNIX." User's Manual Unix Operating System Vol 2. Murray Hill, NJ: Bell Laboratories,1980, 108-120.

Kernighan, B. W., and Pike, R. The Unix Programming Environment. Englewood Cliffs, NJ: Prentice-Hall, 1984.

Laursen, A. L. "GPSI : An Expert System to Aid in Program Debugging." M.S. Thesis, University of Illinois, Urbana, Il. 1981.

Lukey, F. J. "Understanding and Debugging Programs." International Journal of Man-Machine Studies 12,2, (1980), 189-202.

Maranzano, J. F. and Bourne S. R. "A Tutorial Introduction to ADB " in Wollongong Group Edition VII Workbench : Programmer's Manual Volume 2. Oceanport, NJ: Perkin-

Elmer Corporation, 1982 1-28.

Mayer, R. E. "The Psychology of How Novices Learn Computer Programming." Computing Surveys 13, 1 (1981), 121-141.

Mc Cracken D. L. and Akscyn R. M. Experience with the ZOG Human-Computer Interface System. Pittsburgh, PA: Carnegie-Mellon University, 1984.

Pierson, J. K. and Horn J. A. AEDS Journal 3, 17 (1984), 53-60.

Schwarz, E., Beldie I., Pastoor, S. "A Comparasion of Paging and Scrolling for Changing Screen Contents by Inexperienced Users." Human Factors 25, 3 (1983) 279-282.

Simpson, H. " Style Guide for Program Design." in Walker, D. F. and Hess, R. Instructional Software : Principles and Perspectives for Design and Use Belmont, CA: Wadsworth Publishing Co.,1984, 130-142.

Steffen, J. L. "Experience with a Portable Debugging Tool." Software-Practice and Experience 14, 4 (1984) 323-334.

Stefik, M., Aikens, J., Balzer, R., Benoit, J., Birnbaum, L., Hayes-Roth, F., and Sacerdoti, E. "The Organization of Expert Systems, A Tutorial. " Artificial Intelligence 18, 2(1982) 135-173.

Steinberg, E. L. Teaching Computers to Teach. Hillsdale, NJ: Lawrence Erlbaum Associates, 1984.

Weiser, M. "Programmers Use Slices When Debugging." Communications of The ACM 25, 7 (1982) 446-452.

APPENDIX A

FRAME BASED REFERENCE PACKAGE MODULE SCHEMATICS

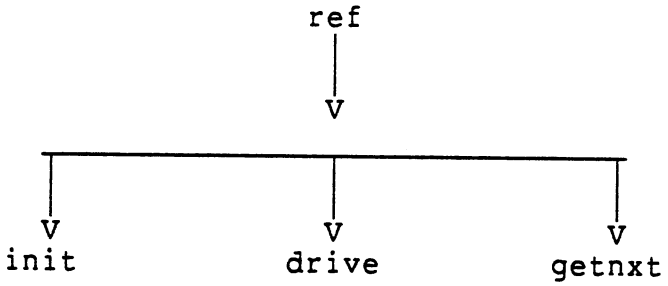

Figure 5. Major Modules for User Package

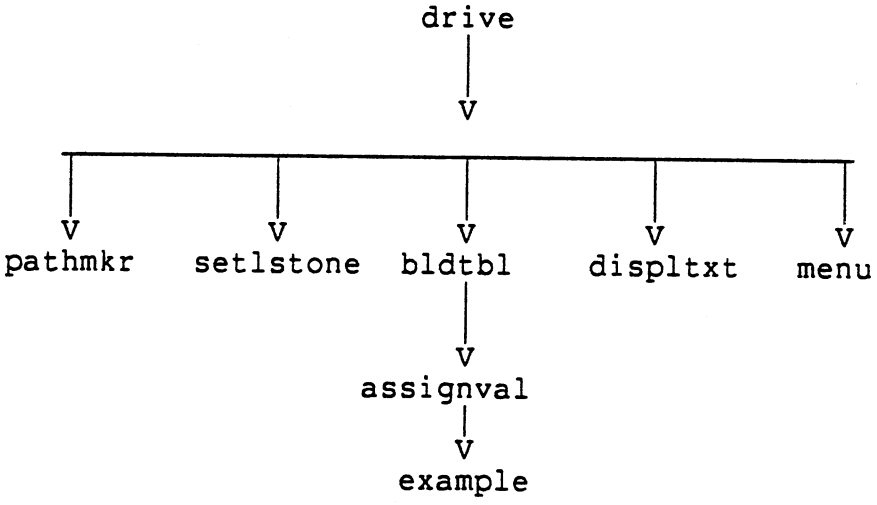

Figure 6. Display Modules

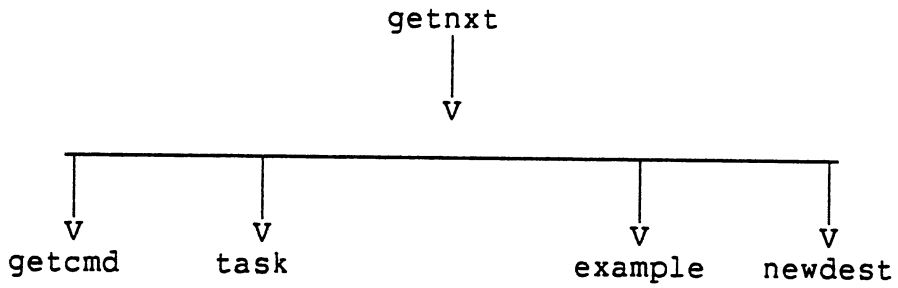

Figure 7. Execution Modules

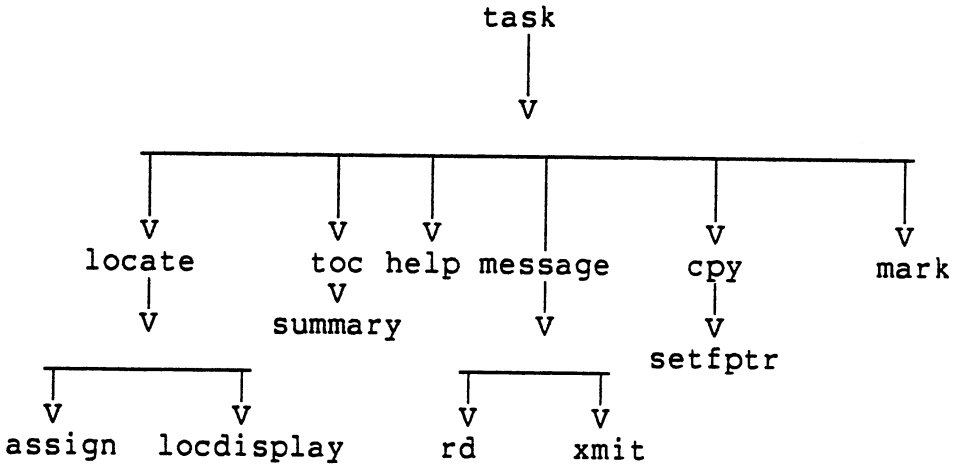

Figure 8. Task Modules

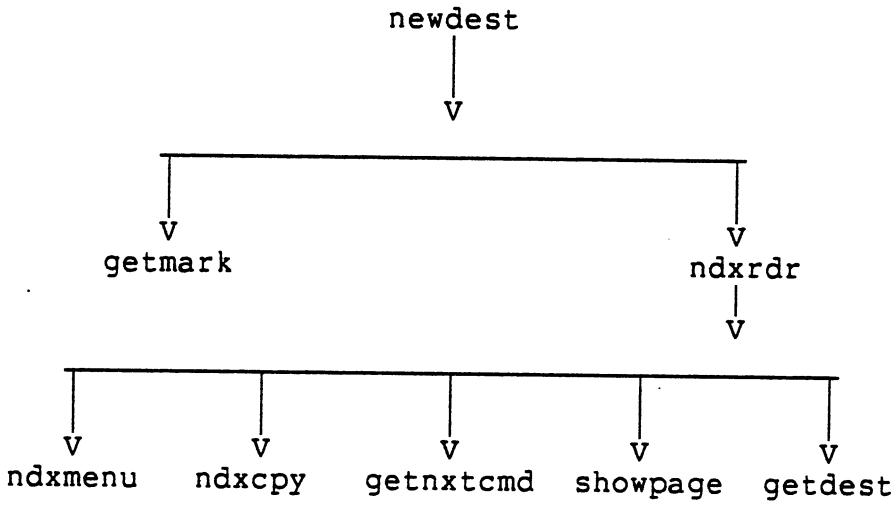

Figure 9. New Destination Modules

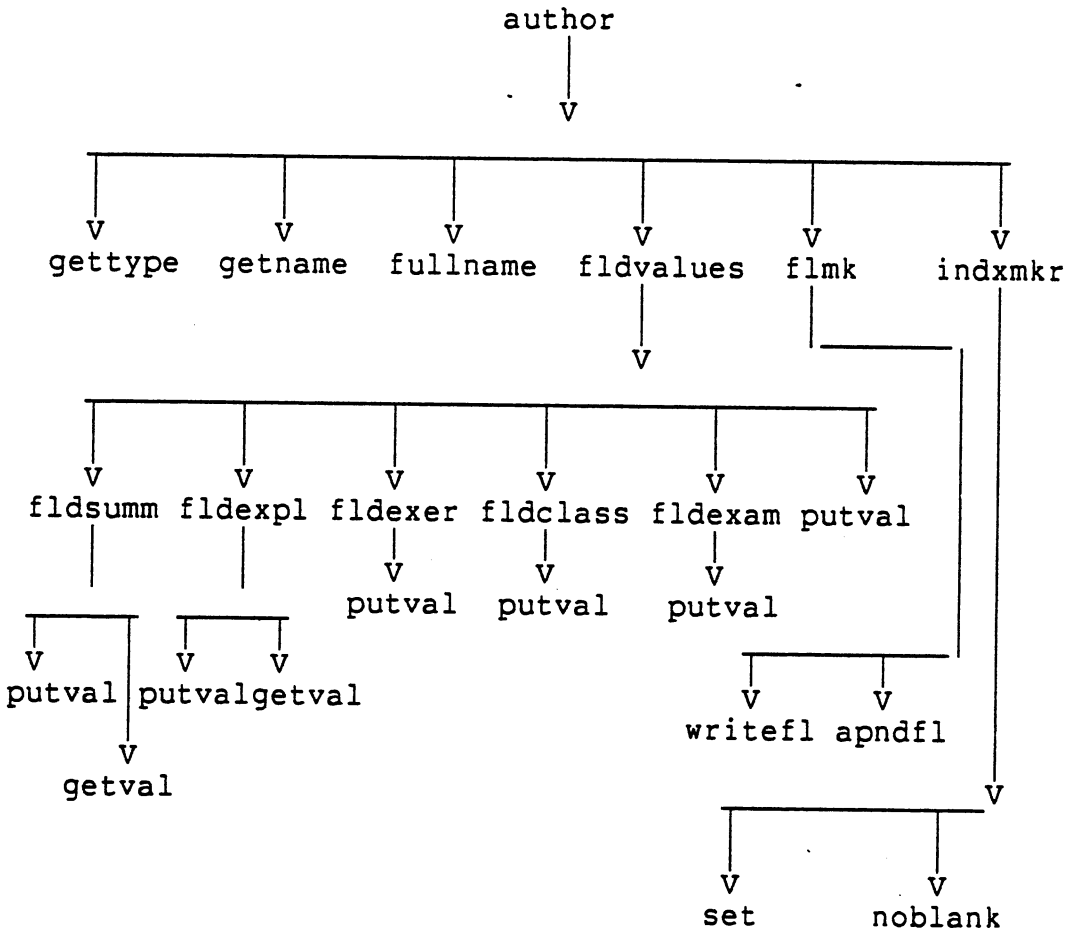

Figure 10. Authoring System Modules

APPENDIX B

FRAME BASED REFERENCE PACKAGE MODULE CATALOG

Major Sections of the User Code

The user code breaks into three parts: the initialization section to set up the beginning frame, the display section, and the execution section to execute user entered commands and return the next destination.

CONCEPTUAL ORGANIZATION

MAIN
  I. INITIALIZATION
 II. DISPLAY
III. EXECUTION

MODULE EQUIVALENT

reference
  I. init
 II. drive
III. getnxt

PROGRAM NAME: ref

PURPOSE: main routine for reference package

CALLED FROM: NA

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: init - initialize
drive - process frame
getnxt - next frame

VALUES RETURNED: none

LOGIC OVERVIEW:

```
Get initial frame name (init)
While (frame name not NULL)
    display frame (drive)
    get the next frame name (getnxt)
```

I. Initialization

The initialization section does all the required set up to start the reference package.

PROGRAM NAME: init

PURPOSE: 1) Set up global table locations
2) Get initial location
3) Return location value

SUB PROGRAMS CALLED: none

CALLED FROM: ref

PARAMETERS PASSED: none

VALUES RETURNED: frstr - pointer to name of first frame

LOGIC OVERVIEW:

Get starting frame name
Return it to calling routine

II. DISPLAY SECTION

The display section has two major functions. First it must build a table that specifies all the frames's connections to adjacent frames and members of the frame set. It also displays the frame and the appropriate menu.

DISPLAY SECTION MODULES

II. drive
- A. pathmkr
- B. setlstone
- C. bldtbl
  1. assignval
  2. example
- D. displtxt
- E. menu

PROGRAM NAME: drive

PURPOSE: 1) set up table for this frame
2) present text

SUB PROGRAMS CALLED: bldtbl - build table
displtxt - displ frame
setlstone - set
last frame name
menu - display menu
pathmkr - full path
name for file

CALLED FROM: ref

PARAMETERS PASSED: name - of frame to be processed

VALUES RETURNED: name - of frame or NULL

LOGIC OVERVIEW:
make a full pathname for passed parameter name
open file
if valid
record name of this and past frame (setlstone)
build table of connecting frames (bldtbl)
display text of this frame (displtxt)
else
abort for bad data
return validity check

PROGRAM NAME: pathmkr

PURPOSE: convert a local name to full path

CALLED FROM: drive,cpy,locate,ndxmkr,rd

PARAMETERS PASSED: ptr - pointer to name

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW:
construct and return complete pathname
for given file name

PROGRAM NAME: setlstone

PURPOSE: To reset lastone and thisone, the present and last visited frames.

CALLED FROM: drive

PARAMETERS PASSED: name - pointer to present frame name

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW: Store last frame visited
Store the present frame

PROGRAM NAME: bldtbl - build table

PURPOSE: extract field values from file header and assign them to global table

CALLED FROM: drive

PARAMETERS PASSED: fp - file pointer

SUB PROGRAMS CALLED: assignval - assign values for field

VALUES RETURNED: good - flag of good data

LOGIC OVERVIEW:

```
while( the end of table marker and end of file not found)
    if line has a field
       extract value
       assign to global field variable (assignval)
    get next line
if there was no end of table marker
    signal error
return validity signal
```

```
PROGRAM NAME: assignval

PURPOSE: to assign value to a field of
         the frame table

CALLED FROM: bldtabl

PARAMETERS PASSED: cln - location of colon
                   diff - length of string
                   flg - value for switch
                   clssnm - class name

SUB PROGRAMS CALLED: example - extract number
                       and get right example

VALUES RETURNED: none

LOGIC OVERVIEW:
        switch(field name)
          assign value to field
          concatenate end of string marker to value
          case example:
              extract number of example set (example.c)
              access that member of example array
              assign value to field
              concatenate end of string to value string
```

```
PROGRAM NAME: example

PURPOSE: given a number, find appropriate example pointer

CALLED FROM: assignval, getnxt

PARAMETERS PASSED: commnd - a number

SUB PROGRAMS CALLED: none

VALUES RETURNED: pointer to example field

LOGIC OVERVIEW: convert example set number to integer
                set a pointer to the this location
                return this pointer
```

PROGRAM NAME: displtxt

PURPOSE: display frame text on standard output

CALLED FROM: drive

PARAMETERS PASSED: flptr - pointer to file

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW:
```
clear screen
while (not end of frame)
   print line of text
   increment line counter
while (line counter less than menu start)
   print blank lines
```

PROGRAM NAME: menu

PURPOSE: present a frame based menu

CALLED FROM: drive

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW:
```
print all menu items that are in every frame
for other commands
    if (command is valid for frame)
       print menu corresponding menu item
```

III. EXECUTION SECTION

The execution section captures, interprets, and executes user entered commands. The commands fall into two major categories: those that do not result in a new frame being visited and those that do.

III. EXECUTION SECTION
   A. CAPTURE COMMAND
   B. EXECUTE TASKS NOT GENERATING A NEW FRAME
   C. EXECUTE TASKS GENERATING A NEW FRAME
MODULE EQUIVALENT

III. getnxt
   A. getcmd
   B. task
   C.
      1. example
      2. newdest

```
PROGRAM NAME: getnxt

PURPOSE: 1) interpret the next commands
         2) invoke server routines
         3) invoke new destination commands

SUB PROGRAMS CALLED: example - process esample
                     newdest - new destination
                     getcmd - get command
                     task - handle utilities

CALLED FROM: ref

VALUES RETURNED: return next destination

LOGIC OVERVIEW:
      while (no new destinations are generated)
         get the command (getcmd)
         if (command is default command)
            destination = name of next frame
         else if (command is quit)
            destination = NULL
         else if (command invokes a service routine)
            call service module (task)
            if (appropriate)
              destination = name of present frame
         else if (command refers to example frames)
              destination = name of example frame (example)
         else
              destination = new destination handler (newdest)
```

A. CAPTURE COMMAND

PROGRAM NAME: getcmd

PURPOSE: get command from terminal

CALLED FROM: getnxt,locate,rd,xmit,message

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: none

VALUES RETURNED: command

LOGIC OVERVIEW:

```
prompt for command
if (command not default command)
   process until the command line is complete
return command
```

B. TASKS THAT DO NOT GENERATE A NEW FRAME

The functions that do not involve a change in presentation frame of the frame set locate the frame set with respect to the whole, send or receive messages, copy frames, mark frames for future reference, present the table of contents, and present explanations for the commands.

MODULAR EQUIVALENT

B. task
   1. locate
      a. assign
   2.message
      a. rd
      b. xmit
   3. cpy
      a. setfptr
   4. mark
   5. toc
   6. help

PROGRAM NAME: task

PURPOSE: perform tasks that do not require a new frame

CALLED FROM: getnxt

PARAMETERS PASSED: command - to be interpreted

SUB PROGRAMS CALLED:

1) locate
2) message
3) cpy
4) mark
5) toc(table of contents)
6) help

VALUES RETURNED: none

LOGIC OVERVIEW:

```
switch(command)
  case locate command
     call (locate) to show present location
  case message command
     call (message) to send or read a message
  case copy command
     call (cpy) to copy present frame
  case mark command
     call (mark) to put present frame on mark stack
  case toc command
     call (toc) to present table of contents
  case help command
     call (help) to present explanations of menu commands
  default
     print error message
```

PROGRAM NAME: locate

PURPOSE: To provide the path from the present frame
to the beginning.

CALLED FROM: task

PARAMETERS PASSED:none

SUB PROGRAMS CALLED: assign, locdisplay

VALUES RETURNED: none

```
LOGIC OVERVIEW: clear the screen
                set name to present file
                while (frame not prior to first frame)
                  open file name
                  extract values for frame name, subject,
                    type, and prior frame on path
                  store frame name subject and type
                  close frame name
                  set name to predecessor frame name
                display stored frames(locdisplay)
```

PROGRAM NAME: assign

PURPOSE: extract value from string and assign to field

CALLED FROM: extr, locate

```
PARAMETERS PASSED: pointer - to receiving  string
                   cln - pointer to colon in string
```

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

```
LOGIC OVERVIEW:  extract value from parameter cln
                 copy into pointer
                 add end of string marker
```

PROGRAM NAME: locdisplay

PURPOSE: display the location stack

CALLED FROM: locate

PARAMETERS PASSED: linestack - array of
offsets into stack
place - intial offset

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW: open file of frames
while (not first one)
print frame data
print first in reverse video

PROGRAM NAME: toc - table of contents

PURPOSE: To indicate the contents of the
package at any one time.

CALLED FROM: task

PARAMETERS PASSED:none

SUB PROGRAMS CALLED: assign
pathmkr
summary

VALUES RETURNED: none

LOGIC OVERVIEW: start at first frame
while(not at end)
print topic of frame set
reverse video present
frame topic

PROGRAM NAME: summary

PURPOSE: To note the present summary
name for this frame set.

CALLED FROM: toc

PARAMETERS PASSED:none

SUB PROGRAMS CALLED: assign

VALUES RETURNED: present -- name of summary frame

LOGIC OVERVIEW:
```
while (not found)
   check for summary field
   if (value = dummy)
       set to present name
   else
       use field value
```

PROGRAM NAME: message

PURPOSE: send a message to predesignated party

CALLED FROM: task

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: xmit - send message
rd - read from bulletin board

VALUES RETURNED: none

LOGIC OVERVIEW:
```
clear the screen
prompt for read or write (getcmd)
if (write)
   call (xmit) to send message
else
   call (rd) to read bulletin board
```

PROGRAM NAME: rd - read

PURPOSE: allow user to read bulletin board

CALLED FROM: message

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW:

```
open bulletin board file if it exists
while (not end of file)
     print line
```

PROGRAM NAME: xmit

CALLED FROM: message

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: getcmd

VALUES RETURNED: none

LOGIC OVERVIEW:

```
prompt for destination
set destination
prompt for message
while (not end of message symbol)
   write input line to message file
prompt if want to append file (getcmd)
if (yes)
   append to message file
send message file by means of system utility
        mhmail
remove intermediate files
```

PROGRAM NAME: cpy

PURPOSE: Copy from a frame to a user's
directory under the name frame.copy

CALLED FROM: task

PARAMETERS PASSED: pointer to frame

SUB PROGRAMS CALLED: flptr - pointer past table section
pathmkr - set full path name for file

LOGIC OVERVIEW:

```
set fullpath name (pathmkr)
open source frame file
open receiving file
if (both are valid)
   move source pointer past table area (setflptr)
   get line
   while (not end of file)
      append line to receiving file
      get next line
```

PROGRAM NAME: setfptr -set file pointer

PURPOSE: to move beyond header to begin text.
Prevents copying of header

CALLED FROM: cpy

PARAMETERS PASSED: flptr - pointer to file

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW:

```
get line from file
while (end of table symbol not in line)
      get line of file
return pointer to beginning of text
```

PROGRAM NAME: mark

PURPOSE: To add to the stack containing
the marked frame names

CALLED FROM: task

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW:

```
if (stack is full)
    transmit error message
else
    add present name to stack
```

PROGRAM NAME: help

PURPOSE: display explanations of commands

CALLED FROM: task

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: fullpath

VALUES RETURNED: none

LOGIC OVERVIEW: print explanation page

C. EXECUTE TASKS THAT GENERATE NEW FRAMES

The purpose of these modules is to generate a new frame for display and execution. There are two major modules involved. One that handles determining example frames and another that provides a catch all for other types. The new destinations come from from the frame's global table with two exceptions. The first involves recovering the name of a marked frame. The second involves going throught the index frame for direct access to the frame.

The index is really a small version of the main program although its frames are stored in a different format. It constitutes a set of frames by itself with its own menu offering a copying utility, display, and going to the next destination.

CONCEPTUAL PROGRAM

C. TASKS RETURNING A NEW DESTINATION
   1. EXAMPLE HANDLER
   2. OTHER COMMANDS
      a. FROM GLOBAL TABLE
      b. FROM MARKED STACK
      c. THROUGH INDEX
         1. INDEX MENU
         2. INDEX COPY
         3. GET NEXT COMMAND
         4. DISPLAY AN INDEX PAGE
         5. EXTRACT A FRAME FROM PAGE

MODULAR EQUIVALENT

C.
1. example
2. newdest
   a.
   b. getmark
   c. ndxrdr
      1. ndxmenu
      2. ndxcpy
      3. getnxtcmd
      4. showpage
      5. getdest

PROGRAM NAME: example

PURPOSE: given a number, find appropriate example set pointer

CALLED FROM: assignval, getnxt

PARAMETERS PASSED: commnd - a number

SUB PROGRAMS CALLED: none

VALUES RETURNED: pointer to example field

LOGIC OVERVIEW: convert example set number to integer
use integer to reference global array
return address into array

```
PROGRAM NAME: newdest

PURPOSE: Given commands, will return
         the pointer to name of new frame.

CALLED FROM: getnxt

PARAMETERS PASSED:cmd

SUB PROGRAMS CALLED: none

VALUES RETURNED: pointer to frame name

LOGIC OVERVIEW:
      switch(command)
        case backward
           destination = global previous
        case marked frame
           destination = (getmark) from top of stack
        case index
           destination = name returned from (ndxrdr)
        case previous frame
           destination = name of previously visited frame
        case summary
           destination = name of summary for frame set
        case explanation
           destination = name of explanation frame
        case exercise
           destination = name of exercise frame
        else
           destination = destination
                  return desintation
```

PROGRAM NAME: getmark

PURPOSE: To retrieve from the top of the mark stack,
the name of its frame

CALLED FROM: task

PARAMETERS PASSED: none

SUB PROGRAMS CALLED:none

VALUES RETURNED: backward - a pointer
to the string on the top of the stack.

LOGIC OVERVIEW:
```
if(mrked frame stack is empty)
    signal error
else
    pop name from stack
return name or NULL
```

PROGRAM NAME: ndxrdr

PURPOSE: To present the index frame with
its commands

CALLED FROM: drive

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: getnxtcmd
showpage
getdest
ndxmenu
ndxcpy

VALUES RETURNED: newdest-name of frame
as next destination

```
LOGIC OVERVIEW:
      get name of index file
      set full path name for it (pathmkr)
      try to open
      if (open)
         while (no new destination)
            display one page of index (showpage)
              with lines numbered
            set command choice to bad
            while (choice is bad)
               prompt for command (getnxtcmd)
               switch (command)
                 case previous frame
                   new destination = previous frame
                 case quit
                   new destination = NULL
                 case copy
                   copy index (ndxcpy)
                 case back
                   display previous index page
                 case default
                   display next page of index
                 case number
                   new destination =
                    extract name from line of
                    this number (getdest)
                 default
                    signal bad input
```

MODULE SECTION FOR AUTHORING

These programs are used to write frames for use by the user package. They operate very simply. They create a file for writing and query the user for the table set up. The user has the option of writing his own text or appending already written text to the just created table data.

CONCEPTUAL REPRESENTATION

WRITE FRAME
- A. GET TYPE OF FRAME
- B. GET NAME OF FRAME
- C. CONSTRUCT FULL NAME
- D. GENERATE TABLE
- E. PRODUCE TEXT
  - 1. WRITE TEXT
  - 2. APPEND TEXT
- F. INCORPORATE NEW FRAME INTO INDEX

MODULAR REPRESENTATION

author
- A. gettype
- B. getname
- C. fullname
- D. fldvalues
  - 1. putval
- E. flmk
  - 1. writefl
  - 2. apndfl
- F. indxmkr

PROGRAM NAME: author

PURPOSE: construct a frame

CALLED FROM: NA

PARAMETERS PASSED: NA

SUB PROGRAMS CALLED: gettype
fullname
fldvalues
flmk
getname

VALUES RETURNED: none

LOGIC OVERVIEW: find frame type (gettype)
get frame name (getname)
add suffix to name (fullname)
fill table (fldvalues)
add text (flmk)

PROGRAM NAME: gettype

PURPOSE: find what type of frame is being constructed

CALLED FROM: author

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: none

VALUES RETURNED: choice - type of frame

LOGIC OVERVIEW: prompt for choice of type
read choice
return choice

PROGRAM NAME: getname

PURPOSE: get name of frame being made

CALLED FROM: author

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: none

VALUES RETURNED: name - of frame

LOGIC OVERVIEW: prompt for name of frame
return it

PROGRAM NAME: fullname

PURPOSE: add proper suffix to name

CALLED FROM: author

PARAMETERS PASSED: name - of frame
kind - type of frame

SUB PROGRAMS CALLED: none

VALUES RETURNED: name - with suffix added

LOGIC OVERVIEW: check for type
add suffix to name
return new name

PROGRAM NAME: fldvalues

PURPOSE: fill in all field values for table
at initial part of frame

CALLED FROM: author

PARAMETERS PASSED: fp - pointer to file

SUB PROGRAMS CALLED: putval - fill in field
fldsumm - fill in summary field
fldexpl - fill in explanation field
fldexam - fill in example field
fldexer - fill in exercise field
fldclss - fill in class field

VALUES RETURNED: none

LOGIC OVERVIEW: for each field in table
fill in value

PROGRAM NAME: putval

PURPOSE: fill in field from frame's table

CALLED FROM: fldvalues

PARAMETERS PASSED: number - field number
flptr - pointer to file

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

```
LOGIC OVERVIEW:  read string into value
                 if (empty)
                   fill in default value
                 write value to file
```

```
PROGRAM NAME: fldsumm

PURPOSE: to fill the summary field in
                    frame table

CALLED FROM: fldvalue

PARAMETERS PASSED: type - of frame
                   number - of field label
                   flptr - pointer to file

SUB PROGRAMS CALLED: putval - in field
                     getval - from field

VALUES RETURNED: none

LOGIC OVERVIEW: if type is summary
                  enter dummy value
                  examine previous frame type
                  if(summary)
                    use its name
                  else
                    use name of summary in
                    its table
```

PROGRAM NAME: fldexpl

PURPOSE: to fill the explanation field in
frame table

CALLED FROM: fldvalue

PARAMETERS PASSED: type - of frame
number - of field label
flptr - pointer to file

SUB PROGRAMS CALLED: putval - in field
getval - of prev field

VALUES RETURNED: none

LOGIC OVERVIEW:
```
switch( type of frame)
  case summary
       prompt for expl name
  case explanation
       enter dummy value
  case example or exercise
       examine previous frame
         if(prev = summary)
            expl = dummy
         if(prev = expl)
            expl = prev
         if( other)
            expl = expl(prev)
```

```
PROGRAM NAME: getval

PURPOSE: extract value from frme table and
         put into location of pointer

CALLED FROM: fldsumm, fldexpl

PARAMETERS PASSED:  number - of field name
                    flptr - pointer to frame
                    ptr - pointer to output

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW:
          while( not done, get next line)
               check line for field name
               if (desired one)
                  extract value
                  save value at pntr
                  set done flag
```

```
PROGRAM NAME: fldexam

PURPOSE: to fill the example field in
         frame table

CALLED FROM: fldvalue

PARAMETERS PASSED: type - of frame
                   number - of field label
                   flptr - pointer to file

SUB PROGRAMS CALLED: putval - in field

VALUES RETURNED: none

LOGIC OVERVIEW: if type is summary or
                explanation
                  prompt for need for
                  examples
                if OK for examples
                  invoke putval to put value
                  in field
                else
                  invoke putval for dummy
                  values
```

PROGRAM NAME: fldexer

PURPOSE: to fill the exercise field in
frame table

CALLED FROM: fldvalue

PARAMETERS PASSED: type - of frame
number - of field label
flptr - pointer to file

SUB PROGRAMS CALLED: putval - in field

VALUES RETURNED: none

LOGIC OVERVIEW: if type is summary or expl.
prompt for exercise field
invoke putval to put value
in field

PROGRAM NAME: fldclass

PURPOSE: to fill the class field in
frame table

CALLED FROM: fldvalue

PARAMETERS PASSED: type - of frame
number - of field label
flptr - pointer to file

SUB PROGRAMS CALLED: putval - in field

VALUES RETURNED: none

LOGIC OVERVIEW: switch on type
fill field with proper
label for type

PROGRAM NAME: flmk

PURPOSE: to add text to frame

CALLED FROM: author

PARAMETERS PASSED: flptr - pointer to file

SUB PROGRAMS CALLED: writefl - write to file
apndfl - append to file

VALUES RETURNED: none

LOGIC OVERVIEW:
```
prompt for choice
if (choice = 1)
   write own text (writefl)
else if (choice = 2)
   append text (apndfl)
else
   signal error
```

PROGRAM NAME: writefl

PURPOSE: write text for new frame

CALLED FROM: flmk - file make

PARAMETERS PASSED: fp - pointer to file

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW:
```
prompt for text line
while (more lines)
   write to file
   prompt for next line
```

PROGRAM NAME: apndfl

PURPOSE: append existing file to passed
parameter filenm

CALLED FROM: flmk - file make

PARAMETERS PASSED: filenm - file name

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW:
```
prompt for name of file
  to append
construct system command
  line
do system command
```

PROGRAM NAME: indxmkr - index maker

PURPOSE: routine to automatically create
an index from all files ending in "txt"

CALLED FROM: NA

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: set - get title field

VALUES RETURNED: none

LOGIC OVERVIEW:
```
Put all files ending in ".txt"
  into a temporary file temp.jorp
open temporary file temp.jorp
open temporary file temp2.jorp
while (another line in temp.jorp)
   get file name from temp.jorp
   extract subject string (set)
   separate string by comma into subjects
   eliminate leading blanks (noblank)
   write subjects and frame  name and type
       to temp2.jorp
sort file temp2.jorp into index file
remove temporary files
```

PROGRAM NAME: set

PURPOSE: extract field values from header

CALLED FROM: indxmkr - index maker

PARAMETERS PASSED: flptr - pointer to file
titlptr - array for title

SUB PROGRAMS CALLED: extr - extract title

VALUES RETURNED: none

LOGIC OVERVIEW: extract title values from string (extr)
store in global field "titl"

PROGRAM NAME: extr - extract

PURPOSE: find and assign from table the
title and class values to global fields

CALLED FROM: set

PARAMETERS PASSED: fp - pointer to file

SUB PROGRAMS CALLED: assign - assign values

VALUES RETURNED: none

LOGIC OVERVIEW:

```
get line from file of fp
while( search not done and no end of table marker)
   check for and extract title  and type fields
   asssign to global fields "titl" and "clss"
return
```

PROGRAM NAME: noblank

PURPOSE: remove leading blanks in string
pointed to by stringptr

CALLED FROM: indxmkr

PARAMETERS PASSED: stringptr - pointer to a character string

SUB PROGRAMS CALLED: none

VALUES RETURNED: none

LOGIC OVERVIEW:
```
while(stringptr points at a blank)
     increment stringptr
return stringptr
```

PROGRAM NAME: indexer

PURPOSE: to invoke indxmkr : index maker
independently

CALLED FROM: N___A

PARAMETERS PASSED: none

SUB PROGRAMS CALLED: indxmkr

VALUES RETURNED: none

LOGIC OVERVIEW: invoke the index maker

APPENDIX C

USER'S GUIDE TO THE FRAME BASED REFERENCE PACKAGE

Getting Started

To get started in the system type "/u/tjj/lthesis/text/ref". You will be presented with an introduction frame with a menu at the bottom of the page. If this is your first usage, you might use the "e" key to go into some orientation frames that contain operating instructions about this on-line reference package. Keep hitting the return or enter key as you finish reading the text on that page. Eventually, you will return to the introductory frame from which you started. And now you are ready to go. What follows is a fuller explanation and background that is provided on-line and might be worth some study time.

Background

You are now in the on-line reference package. It provides a concept different from most on-line aids in that it attempts to provide in depth explanation to whatever degree required. The system presents materials by pages which in the terminology of the system are referred to as frames. Let us now examine the frame types that you could visit as you operate the reference.

Summary: This frame type presents a brief overview of a topic. If that provides you with adequate information, you need go no farther. When browsing or traversing this package, you will be travelling along summary frames as you go from topic to topic. The summary frames are the connections for the whole package.

Explanation: If you feel that the information presented in the explanation frame is incomplete and unclear, you may visit the explanation frame(s) provided with this topic. Be aware of the fact that all topics do not have explanation frames. This is true if the topic is simple, and more explanation than that provided in the summary frame is not deemed necessary. There may be more than one explanation frame provided for this topic. Remarks in the text will lead you to the other explanation frames provided with this topic. Often throughout this presentation, we will use the term frame sets to indicate the possibility of multiple frames of a particular type concerning a particular topic.

Example: For every topic and many subtopics, examples are present to illustrate concepts. If you wish for more concrete information than that provided by the textual information, you may visit example frames that illustrate these concepts. Generally,

there are a set of example frames for one topic rather than just one.

Exercise : If you are serious in knowing concepts, it is suggested that you perform the exercises. In format, the exercises are similar to the examples. A situation is presented and you are asked to analyze it and construct an answer. The next frame presents an answer that you may compare to your own. The exercises are not difficult and intended for reinforcement rather than extending concepts.

Index : This special frame allows you to directly access explanation and summary frames. When you invoke this frame, the topics of all explanation and summary frames are presented in alphabetical order and numbered. Entering the number associated with the topic you desire will cause that frame to be presented. Exercise and example frames must be accessed from the associated explanation or summary frame.

Message : It may often be advantageous to communicate about the package. You may find that the presentation leaves questions unanswered, the text may be inaccurate, or you may have run across situations that are not covered in the message package. Fine; one of the design principles of this package was that it be easy to modify and have the ability to

grow to handle new situations. The direction and type of growth depend upon feedback from the user population. To facilitate communication, a message frame has been introduced that allows the user to send messages to the people connected with the content of the package. This topic will be covered in greater detail with the message command later in this manual.

Communication is a two way street. The writers of the package may want to communicate with the general population. You can read their messages in the bulletin board accessed by using the read command within the message frame. They may even pass along user comments to the general population this way.

Dummy: This is the error frame. If you enter an inappropriate command, you will enter this frame which is instantly recognizable by containing an error statement with directions on how to exit this frame.

Operating Instructions

Operating the package is extremely simple. Once you get on the system, you are presented with some text and a menu near the bottom portion of the frame presentation. The menu contains all the allowable command options available from this frame. You simply choose one and type the appropriate symbol or number followed by return or with the default option, just type return by itself. The only thing you need to is what the commands mean and that is what follows this section.

Commands

h - This is the help command and it may be used from any of the information frames. It describes all the commands in the package.

l - This is the locate command and returns a picture of where you are with regard to the beginning frame of the package. This routine starts with the beginning frame in the package and works forward to the frame you requested it from. It presents the title, topic, and type of each frame along the path. For terminals with the capability, the information for the last frame (your present location) is highlighted.

t - This command presents the table of contents of all the topics presently in the package. On terminals

so defined, the topic that originated this command will be highlighted. For most purposes, this command may be more useful than the locate command for general orientation within the package. What must be noted here is that the table of contents does not list all frames and their topics; it lists only the topics of the summary frames.

m - This invokes the message routine. First, you are presented with the choice of going into the read or transmit mode. In the read mode, invoked by the command "r" while in the message frame, the bulletin board is presented.

The transmit mode works somewhat differently. It is invoked by the command "w" from the message frame. You are then presented with a choice of recipients from the people who are involved with the reference package. At present, these are the instructor, teaching assistant, and system maintainer designated by the numbers "1", "2", or "3" respectively. You then enter one of these numbers and are then prompted to enter your message. You write as much as you desire and end the message by typing a period, "." as the first character on the following line. This will cause your message to be stored.

The next prompt will ask whether you wish to append a file to the message thus far produced.

Often you may have questions or suggestions relating to a particular program. This is the point where you include the name of this program which will be appended to your previous message and sent. At this point you are returned to the main program.

c - This command appends a copy of the present frame to a file in your directory called frame.copy. If frame.copy does not exist, it will be created. This command becomes a handy way of saving the most relevant material you find during your usage of the reference package and saves you from having to take notes on the material as you study it.

k - During a session, you may wish to mark a frame for future reference - one that you wish to return to in the future after you have finished browsing. The command "k" will allow you to do that by simply using this command when prompted in a frame. This command has limitations that you should be aware of. Once a marked frame has been revisited, it is no longer marked and cannot be called up in this manner again unless it is remarked. Frames are visited in the reverse order that they are marked. The first frame marked cannot be visited until all the other marked frames have been visited. If this is unsatisfactory for your usage, then

consider using the index to visit frames directly when desired. Up to nine frames may be marked at one time. Once this point has been reached, some frames must be removed from the stack of marked frames by revisiting them.

g - This command is the opposite of k and causes you to restore marked frames in reverse order of marking. This means that the most recently marked frame will be the first one restored.

default - Default means that instead of entering a command followed by a return or enter, you just enter the return or enter alone. What is presented is the next frame along the path. Except in the case of the last item in a similar set, the next item is of the same type as the present item. For instance, if a topic has several explanation frames, then each time the default key is used, the next explanation frame is visited. This holds for the summary, example, and exercise frames as well. The question becomes : what is the default for the last frame in a set? This default is the frame that invoked the initial member of this set. In our example, the default destination for the last explanation frame of the set is the invoking summary frame. If our example were the last example frame of an example set then the default would be

the explanation or summary frame which invoked it, whichever is appropriate. The only exception to this rule is the summary frame set. Going beyond the last summary frame will cause you to find yourself in the error frame. This is intentional in order to signal the end of frame condition.

b - This command (back) presents the previous frame on this frame path. It is identical to the default command, but in the opposite direction. It follows sets to their beginning point, to their invoking frame, and ultimately to the beginning of the package. At the summary level, exceeding the limits will again enter the error frame.

p - This command visits the previously visited frame. This command has several valuable uses. It is useful for departing the error frame and is very good for flip - flopping back and forth between two frames.

i - Invoking this command will cause the index frame to be visited. From these frames, one may directly visit any indexed frame.

You are presented with eighteen lines of the index entries. Each line contains the topic, name, and type of the frame. Only summary and explanation frames are included in the index. The user is presented with a menu that allows the rest of the

index to be perused eighteen lines at a time (the default command), or a backward traversal to previously visited index pages (by means of the "b" command), the index to be copied (the "c" command), to quit with a "q", or return to the previous non- index frame. You have one other option that makes the index frame unique. By typing one of the entry numbers, that frame is directly invoked without any reference to an intermediate frame or path.

q - This is the quit command and causes you to leave the reference package.

s - This command is available in the explanation, exercise, and example frames. It restores the summary frame for this topic. This command is illegal from a summary frame though.

e - This command when displayed in a summary frame invokes the first explanation frame for that topic (if such a frame exists for this topic). If invoked in an example or exercise frame, it returns to the invoking explanation frame. This command is not valid when issued from an explanation frame.

x - This command causes the exercise frame associated with the invoking frame to be visited.

number - Numbers invoke the first member of the num-

bered example set involved with the invoking frame. The text of the invoking frame will mention which numbers are appropriate for this frame. Using the mentioned number will begin the frame set that matches that number.

That concludes the instruction portion of the operating manual; what follows is a step by step example that uses a number of the commands in order to give a flavor of how the package works. A further note might be in order concerning the error frame. All roads lead nowhere except three. If "p" is invoked as the first command, then you are returned to the previous frame. If any erroneous commands are invoked, there is a good chance that "p" will no longer work. You may still have the option of going to the index frame, a previously marked frame, or simply quit at this point.

As a summary and reference, here are the available commands:

b - back

c - copy

e - explanation frame

g - go to marked frame

i - index

k - mark

l - location

m - message

p - previous

q - quit

s - summary frame

1 - example set 1

2 - example set 2

9 - example set 9

<return> - next frame

t - table of contents

x - exercise

A Sample Terminal Session

We will proceed step by step through a terminal session. We assume that the user has logged in and accessed the package by whatever means is required. At present this is "/u/tjj/lthesis/text/ref". The user will now see the introductory frame illustrated in figure 11.

For an introduction and operating instructions, please
press an "e" followed by return. Otherwise, the standard
keys will move you on.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
please enter next command:

Figure 11. Introductory Frame

As may be seen, the user has entered the default command, "return", to proceed along the summary frame path. Figure 12 shows the next frame with the command "e" entered.

Compiler Introduction

The following section deals with the interpretation of compiler error
messages. To enter the section please follow the default, otherwise
more information is available in the explanation section.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
please enter next command:e

Figure 12. Compiler Summary Frame

The use of the command "e" in the previous frame invokes the the explanation frame for this topic, so the next frame seen will be this explanation frame. You will notice that no command can be seen in the command line. This means that our user has used the default command and since he is in an explanation frame, this means that he will either move to the next explanation frame if there are any or return to the summary frame if there are not.

Introduction

The purpose of the compiler is to convert your source code into something that the computer can execute. In doing so, it must make sure that your code follows the rules of the language. It checks your code for correctness and when it finds a discrepancy it generates a message. Unfortunately, the condition that causes an error may not match the message very well. The condition that causes the message may be some distance away from the place that generates the message. Because of these situations, the error messages that are produced may be misleading.

Often debugging involves finding clues about the errors. These error messages are one such clue. The purpose of this section is to add to these clues.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCsSUMreturnNEXT
please enter next command:

Figure 13. Explanation Frame for Compiler Summary Frame

Our example user has come into the next explanation frame of this set. He has chosen to return to summary frame with his choice of "s" to the command line prompt. You might note how the menus change from frame to frame. Only the legal commands from any frame are presented.

TIPS FOR THE STUDENT

1. DON'T TAKE ALL COMPILER MESSAGES SERIOUSLY
   The C compiler is very prone to produce "cascade" error messages. Cascading means that one error may produce a number of messages because of what the original error produced.

2. LOOK FOR CLUSTERS OF ERROR MESSAGES
   Because of cascading, one error may cause several cascade messages in the same line or the same line and one or two following lines to form a cluster.

3. PAY MOST ATTENTION TO THE FIRST ERROR MESSAGE OF A CLUSTER
   More than likely, the messages in a cluster are cascaded and probably are erroneous. Correction of the first condition of a cluster will usually cause the rest to disappear.

4. CHECK THE PRECEDING LINE IF AN ERROR IS NOT APPARENT.
   Sometimes an error not caught on one line will cause another error message to appear on the following line. This is especially true of punctuation errors.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCsSUMreturnNEXT
please enter next command:s

Figure 14. Further Explanation Frame

The user is now back to the summary frame for this topic; it is a frame that we have seen before. This time, the default value is used and we move along the summary chain of frames.

Compiler Introduction

The following section deals with the interpretation of compiler error messages. To enter the section please follow the default, otherwise more information is available in the explanation section.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
please enter next command:

Figure 15. Return to Summary Frame

The student marks this frame for future reference by using the "k" command. Then he moves to the next frame by using the default key.

COMPILER ERROR

<identifier> undefined

Causes
1. identifier not declared this module by error
2. identifier declared in another module but not "#include"d in this module.
3. identifier spelling in declaration and usage do not match (one of them is misspelled).
4. Pointer defines variable and pointer is not initialized.
5. Cascade error
   a) initial quote is missing in a string
   b) semi-colon of preceding line is missing.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
please enter next command:k
please enter next command:

Figure 16. Summary Frame

Here we have another frame visited by default with it also exited by the default command.

```
Expression syntax
     Possible causes
        1. Punctuation error
        2. Undefined variable within the expression
        3. string errors
        4. Incorrect format for expression


cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
please enter next command:
```

Figure 17. Summary Frame

In this frame, we use the command "g" to recall the previously marked frame.

Statement syntax

Possible Causes
1) statement syntax in error .i.e. missing semi-colon
2) an expression within the statement is in error

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
please enter next command:g

Figure 18. Summary Frame With g Command

The marked frame is now revisited. The command that is entered in this frame is "i" which invokes the index frame procedures.

COMPILER ERROR

<identifier> undefined

Causes
1. identifier not declared in this module by error
2. identifier declared in another module but not "#include"d in this module.
3. identifier spelling in declaration and usage do not match (one of them is misspelled).
4. Pointer defines variable and pointer is not initialized.
5. Cascade error
   a) initial quote is missing in a string
   b) semi-colon of preceding line is missing.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
please enter next command:i

Figure 19. Return to Marked Frame

This is an index frame. Note how different it is in appearance to the other frames even to the point that its menu is different. The user takes advantage of the direct access capabilities by deciding to visit item fourteen in the list.

```
1   Bad include syntax              include.txt       summary
2   Can't find include <file>       include.txt       summary
3   compiler intro                  comp.txt          summary
4   compiler intro                  comp2.txt            expl
5   debugging tips                  comp22.txt           expl
6   declaration syntax              dcl.txt           summary
7   expression syntax               expl.txt          summary
8   expression syntax               exp2.txt             expl
9   external definition syntax      xds.txt           summary
10  identifier undefined            undef.txt         summary
11  illegal <pound sign>            illpd.txt         summary
12  illegal <symbol>                illegal.txt       summary
13  illegal indirection             illind.txt        summary
14  illegal structure reference     illstr.txt        summary
15  intro                           intro1.txt        summary
16  introduction                    intro21.txt          expl
17  introduction                    intro22.txt          expl
18  introduction                    intro23.txt          expl

THE AVAILABLE COMMANDS ARE
     default - next index frame  b - previous index frame
     q - quit program p - previous non index frame
     number - to numbered frame  c - copy index
 enter command choice:14
```

Figure 20. Index Frame

This frame was accessed directly by means of the index frame functions.

Illegal structure ref

Possible Cause

Either an attempt has been made incorrectly to declare a structure or there exists a syntax situation where a structure is indicated and none exists. An example of the latter situation is when a pointer points to something that has not been declared.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
please enter next command:l

Figure 21. Summary Frame Visited From the Index Frame

This shows the result of using the locate command. The starting or present frame is the top one. The next item is the frame before this on the path to this frame. This continues until the whole path back to the introduction frame is revealed.

---

subject	name	type
illegal structure reference	illstr.txt	summary
illegal <pound sign>	illpd.txt	summary
non terminated string	stringl.txt	summary
illegal indirection	illind.txt	summary
declaration syntax	dcl.txt	summary
statement syntax	state.txt	summary
expression syntax	expl.txt	summary
identifier undefined	undef.txt	summary
compiler intro	comp.txt	summary
intro	introl.txt	summary

code for type
summary = summary file
expl= explanation file
exm = example frame
exer = exercise frame
enter a return to exit this frame
please enter next command:

Figure 22. Frame Showing Result of Locate Command

After entering a command in locate mode one returns to the previous frame for a new command. Notice that the user entered an "s", and s is not in the menu list of valid commands. This means that the command is invalid.

Illegal structure ref

Possible Cause

Either an attempt has been made incorrectly to declare a structure or there exists a syntax situation where a structure is indicated and none exists. An example of the latter situation is when a pointer points to something that has not been declared.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
Figure 23. Return to Previous Frame After the Locate Command

This frame is the error frame. It is reached by inputting an inappropriate command. That is, the command is "legal" but is not valid for the particular frame in which it is invoked. As the frame indicates, all one has to enter is "p" and the user returns to the previous frame. The user may also escape through the index or by invoking a marked frame. A word of caution, if the user enters another invalid command then the "p" command will not work because the previous frame is also the error or dummy frame. The other commands are still valid though.

If you are reading this you have made an error. To return to your previous file, please hit a "p" and a return.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOC

Figure 24. Error Frame

The user has left the error frame and returned to the previous frame. He now enters a "q" to stop execution of the package and leave it.

Illegal structure ref

Possible Cause

Either an attempt has been made incorrectly to declare a structure or there exists a syntax situation where a structure is indicated and none exists. An example of the latter situation is when a pointer points to something that has not been declared.

cCOPYgGOhHELPiINDXkMARKlLOCmMSGpPREVqQUITtTOCeEXPreturnNEXT
please enter next command:q

Figure 25. Final Frame As User Leaves the Package

APPENDIX D

AUTHORING GUIDE FOR THE ON-LINE REFERENCE PACKAGE

The purpose of the writer is to write frames for a particular reference package. An aid exists called "author" that will aid the writer in performing all the steps necessary to generate the frame. All the writer has to do is respond to the program's queries. The problem is that the writer must understand what is being asked and how to respond to each query. That is the purpose of this manual. Included in this manual is a sample of a query sequence.

The first question concerns the type of frame. The user must enter a number to indicate which type this frame will be. A '1' indicates a summary frame, a '2' indicates an explanation frame, a '3' indicates an example frame, and a '4' indicates an exercise frame. The reason that type is important is that the program generates a suffix for each frame type that is appended to its name which will be the next question asked. The importance of the appended suffix has to do with the automatic index maker which is invoked later in author. The index maker uses the suffix ending to decide which files will be included in the index that will be used by the reference package. For reasons of space,

only the information from summary and index files is included in the index.

Once the name has been generated, the file is opened and the information is now written in. The first set of information defines where the file is in the total framework. This means that it must specify which frame precedes this frame and which frame succeeds it; which frames are accessible from this frame and which are not. The initial text is a table that defines all these things as well as information about the file. The user is queried for this information item by item. Following this paragraph, these items are explained in detail. This task is not as daunting as it may otherwise seem since most frames will require only a small number of fields to be filled in. Author will automatically fill in some of the information that is obvious. Other information may not apply to this situation in which case for the response, you merely enter a "return" without any data. Author will fill in the default value, "dummy", for you. When in the reference package, dummy as a frame name invokes the error frame which informs the user that he has entered an illegal command for that frame and lets him know how to exit.

Frame Fields

titl: This contains the topics of this frame. More than one topic may be appropriate for this frame. Multiple topics must be separated by commas. Leading blanks are

permissible for this field only since they will be stripped off before being stored as a value.

clss: This denotes the type of frame presently processed. You will not be prompted for this information directly. Author prompts for the type in another context and use the information here as well.

next: Next is the default frame destination invoked by a return key being pressed. It contains the name of the next file in this path. An important principle to keep in mind here is the user is to be protected from getting into trouble by pressing the default key. That is, when the user reaches the end of one frame type, he defaults to the frame of the next higher type that called this frame. To make this more concrete, consider each frame as a member of a chain of frames of similar types. Summary frames are one chain type, the explanation frames within a frame set form another chain, the example set another chain, and so forth. These chains may consist of one or more members. The key factor is what happens when the last member of the chain is reached? The answer is that default next is the next higher level chain. In the case of an explanation frame, the next higher level is the summary level. In the case of an example chain, the next higher level is the corresponding explanation frame that started the chain. The explanation frame is the de-

fault for the exercise frame as well. The summary frame is the exception to this rule. When the final summary frame is reached, the default is dummy in order to signal to the user that the end of framework has been reached. Figure 26 is a pictorial representation of how the next field works.

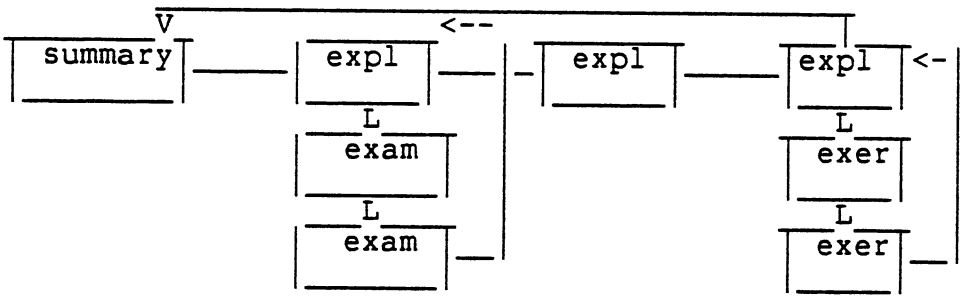

Figure 26. Next Frames for End of Sets

prev: Previous is invoked by the command "b" and denotes the previous frame on the path. It performs the same as next only in the opposite direction. It heads for the beginnings of chains progressively. Its final frame is the intro frame. Previous to this, it runs into the dummy frame again.

summ: The value here is invoked by the "s" command and

denotes the summary frame for this frame set. This field is filled in by author and need not trouble the writer. For information purposes, this value for a summary frame is "dummy" since it is considered an error to try to invoke the summary frame from the summary frame. However, all other frames in the frame set will contain a value for the summary frame.

expl: This frame is invoked by the command "e" and denotes the explanation frame. For the summary frame, this frame is the initial explanation frame in a chain if there are more than one. Except for the summary frame, the value for this field is generated and automatically entered by author.

exm[n]: The first example frame in the chain is called by [n]. [n] stands for a number. For instance the command "1" will call up the value in "exm1", "2" for "exm2" and so forth. Since each example set represents a particular topic, it is possible for a particular explanation frame to several example chains starting at it. In fact, a capability of up to nine example sets is possible from one explanation frame. This is a reminder that this means the possibility of nine sets of examples, not just nine examples. Again, following convention, the example frames themselves fill this field with the value "dummy".

exer: The command "x" will invoke the exercise frame which

is reachable from an explanation frame or a summary frame in an abbreviated frame set. The value for this in an exercise frame is "dummy" as per convention. This is typically coming from the last explanation in a frame set after a topic has been completely explained.

At the end of filling in field values, author asks which option you wish to use to create text. You may write your own text or use text that already exists by appending to the table from that text. You might consider the use of appending for examples and exercises and of writing your own text for summary and explanation frames.

If you choose to write your own text, you should be aware of some guidelines. The text should be tailored to fit within twenty lines. This is a challenge in itself since the ideal to make the presentations modular. This means that each frame should be whole by itself and not depend upon any other frame. The idea here is to prevent the need for constant flipping back and forth between frames to catch the meaning. This will irritate the user and very likely result in lower comprehension rates. There is nothing to present a topic over several frames, but each frame should be a complete subsection of a larger whole. To aid you in writing, you will note that line numbers have been provided for each line of input.

You should be careful of one thing in writing though. The end of input is signalled by inputting only a return on

a line. If you wish to skip lines, please put a blank on each line or else the program will end before you do.

If you wish to append an existing file, the same qualifications exist as for the writing of text. It should fit within twenty lines. The input file may have to be tailored somewhat to meet this requirement. All the program needs to know is the name of the file and it will append it automatically.

Summary and explanation file types will automatically invoke the the index maker to incorporate the new frame into the index. Errors anywhere may be remedied using one of the editors available on the system as would be customary for any file.

To illustrate the authoring system, a sample session follows that adds an example frame to the package. Any uncertainty about any of the procedures should be answered by a review of the pertinent textual reference. It should be emphasized that when a frame is added, this is not the end of the process. The preceding and following frames on the path, if they exist must have their link fields modified to reflect the new member. This is merely a process of using an editor to change the links in these frame files.

Removal of a frame is a manual process. The user must use the vi editor to change the links in the preceding frames. Additionally, if the removed frame is a summary or explanation frame, the index adjusting routine "indexer" should be performed.

Sample Authoring Session

This shows some of the prompts that the authoring system will use to define a frame. Here, the user wants to define an example frame so he enters "3" for type and "toy" for name. The system prompts him for connecting frames which are entered, shown below the prompting line.

```
enter the appropriate number for the correct frame type
a '1' for a summary frame
a '2' for an explanation frame
a '3' for an example frame
a '4' for an exercise frame
please enter next command:3

please enter the name you wish for the file
do not add a type suffix, this will be done automatically
toy

now enter the field values for the table
for any that do not have values enter a return only

enter the topic of the frame
separate multiple topics with commas
demonstration

enter the name of next frame on path
kwd.exm
```

Figure 27. Portion of Prompts for Authoring System

This is a continuation of the authoring system prompts. Note that there are relatively few prompts. Actually author is filling in a number of field values on its own. The author writes some text; the numbers are supplied by the program as a guide. The first line that has nothing on it terminates the input; such a line is line 2.

```
enter the name of previous frame
kywd2.txt

you have a choice of writing your own text
or appending another file of text
to write enter a 1 when prompted
to append, enter a 2
please enter next command: 1

enter text now
there is a limit of 18 lines
even a blank line must have one blank character
or else it is counted as the end of file
1 I hope this is a successful demonstration
2
%
```

Figure 28. Completion of Prompts for Authoring System

This is the result of using the authoring to create text. Note all of the fields that not prompted for containing values. Under different circumstances the user would be prompted for many of these other values. Under this situation author filled in these values.

```
titl:demonstration
next:kwd.exm
prev:kywd2.txt
summ:kwd.txt
exer:dummy
clss:exam
expl:kywd2.txt
exm1:dummy
exm2:dummy
exm3:dummy
exm4:dummy
exm5:dummy
exm6:dummy
exm7:dummy
exm8:dummy
exm9:dummy
# end of table symbol
I hope this is a successful demonstration.
```

Figure 29. Contents of Toy.exm

VITA

Terry Jay Johnson

Candidate for the Degree of

Master of Science

Thesis: A FRAME BASED ON-LINE REFERENCE PACKAGE

Major Field: Computing and Information Science

Biographical:

Personal Data: Born in Independence, Missouri, October 25, 1948, son of Donald L. and Marilyn J. Johnson. Married to Cora E.L. Fillers on June 29, 1972.

Education: Graduated from Ann Arbor High School, Ann Arbor, Michigan, in June, 1972; received Bachelor of Arts Degree in Sociology from University of Michigan in May, 1970; received Master of Library Science Degree in Library Science from Louisiana State University in July, 1978; completed requirements for the Master of Science degree at Oklahoma State University, in December, 1985.

Professional Experience: Medical Corpsman, U.S. Army, July, 1970 to May, 1979; Data Systems Intern, National Security Agency, August, 1979 to June, 1983; Teaching Assistant, Oklahoma State University, August, 1984 to May, 1985.