A KNOWLEDGE BASED TOOL TO AID IN SOFTWARE MAINTENANCE

by

ALBERT L. NICHOL

B.S., Kansas State University, 1984

A MASTER'S THESIS

Submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1987

Approved by:

[illegible]

Major Professor

Table of Contents

LIST OF FIGURES

ACKNOWLEDGEMENTS

The author would like to express his gratitude to Dr. David A. Gustafson for introducing this topic. Without Dr. Gustafson's aid, this project would not have had clear direction.

I would also like to thank Dr. Elizabeth A. Unger and Dr. Austin Melton for their suggestions and input on the thesis as well.

Chapter 1

Introduction, Hypothesis, & Review of Literature

INTRODUCTION

One of the most overlooked parts of the software life cycle is the phase know as maintenance. Recent studies show that maintenance accounts for between forty to eighty percent of the total cost of a software package throughout its complete life [Mar83], [Par82]. Yet, until recently, little attention was given to this phase. Many data processing companies are spending increasing amounts of time and money doing maintenance tasks. As more time is being spent on maintenance of current software, less time can be utilized for the production of new software [Cha83], [Mar83]. In extreme cases companies have stopped new software production just to maintain existing programs [Mar83]. This type of problem demonstrates the need for looking at new and improved methods for doing software maintenance tasks.

The recent attention that has been given to software maintenance has mainly been focused on how to develop maintainable software using better design techniques [Abb83], [Hec83], [Mar83], [Rom85]. This attention to design does not address the problems that maintainers are facing today. Most software that is currently in use was not developed with maintainability in mind. In order to solve the problems that maintainers face today, tools must be developed that will aid with the maintenance phase.

The need for tool development in the maintenance phase of the software life cycle has been largely untouched as an area of study in computer science. The area of software maintenance can be broken into three distinct parts. These parts include: 1) understanding what the program being maintained does and how it does it, 2) doing the actual maintenance task, and 3) the testing the change performed for correctness. It is in this light that I have developed a Maintenance Assistance Tool (MAT1) that will aid maintainers in the learning portion of doing a maintenance task. The tool was developed in the LOOPS programming environment using procedural and object-oriented programming techniques. MAT1 uses a knowledge base to store information about the history of changes and the documentation for the program that is being maintained. Information can be quickly retrieved from this knowledge base to inform maintainers about the program. Some information about the program is displayed graphically for ease of understanding and to help speed the learning time that maintainers must use to become familiar with the program that they are to maintain.

HYPOTHESIS

The need for tools to aid in the maintenance phase has been recognized as one of the areas that can provide an interesting area of study in computer science [Har83], [Kuh85]. The use of expert systems to solve problems is becoming more popular. Expert systems allow computers to acquire information and store it in a knowledge base of information. This knowledge base can then be accessed by the program to help make decisions that help solve the problem at hand [Bro85], [Dym84].

One of the major problems that maintainers face is that they did not write the code for the original program. This means that they must learn about the functions that the program being maintained performs. One of the best sources of information for learning about the program is its documentation. Another source is a history of changes from past maintenance tasks performed on the program [Kuh85], [Mar83].

[Par82]. The learning of what a program does takes a large amount of time. If something can be done to reduce this learning phase of maintenance, some of the problems associated with maintenance can be overcome and the high cost of software maintenance reduced.

My hypothesis is that a knowledge-based tool can provide useful information from distributed documentation including notes about the program being maintained and the history of changes that have occurred. A knowledge-based tool is a tool that gathers information into a knowledge base or data base for use in helping to solve the problem at hand. Distributed documentation and history of changes are separate pieces of information about a program that can be used to provide details about the function of a program and about maintenance that has already been performed on it.

The development of this tool should simplify and speed the organization and extraction of information. By providing information to the maintainer about the program being maintained in an easy-to-read format and/or graphically, MAT1 should be able to shorten the time needed to learn about the program. The shortening of the time needed to learn about a program will help to make maintenance tasks less costly and less error prone.

REVIEW OF LITERATURE

The disproportionate costs associated with software maintenance, as compared to other phases of the software life cycle, peaked my interest in this area. Many companies are spending increasing amounts of time and money in doing software maintenance tasks. Because maintenance is the most costly part of the software life cycle one would think that this area would have been studied in depth. This however, is not the case. The study of software maintenance and tools to aid in maintenance tasks is relatively new. The review of the literature provided interesting insights into the problems associated with software maintenance.

Two main categories of literature were reviewed for this thesis. The first category deals with software maintenance. Software maintenance is used as the theoretical background for development of MAT1. The second category includes expert systems, the use of the LOOPS programming environment, the Lisp programming language, and object-oriented programming techniques. MAT1 was developed in LOOPS which is an environment designed for easy development of expert systems using the Interlisp-D programming language and object-oriented programming.

Software Maintenance

The material dealing with software maintenance is important for providing the theoretical foundation for the derivation of the thesis topic. For many years computer scientists have realized the importance of doing software maintenance tasks [Mar83], [Par82]. Many calls for better maintenance techniques and tools have been made, but little actual work on developing tools to aid in maintenance tasks is documented [Mar83]. Recent work deals with tools that are in a planning stage and so actual working tools that are used by maintainers are few in number and are not available for review [Col85].

Stereotype Associated with Maintenance

The phase called maintenance has had a stereotype of being a beginner's task or a task for persons on their way out [Par82]. It is evident that many persons feel that real programmers do not do maintenance tasks, but instead are reassigned to new development projects leaving novices to do maintenance of programs. Maintenance is often viewed as substandard work and many times it does not command the respect of programmers or management personnel [Par82]. This view of maintenance of software in itself is evidence of the possibility that problems can arise while a maintenance task is being performed. The use of novice programmers to maintain code can cause many problems [Par82].

Problems Associated with Maintenance

The problems associated with software maintenance are extensive. The problems include: 1) bad programming practices which make the software difficult to maintain, 2) novice programmers who are assigned to maintenance tasks often cause as many problems as they solve, and 3) time and money costs which continue to climb [Par82]. The problem of increasing cost is documented extensively in the literature about software maintenance. According to studies presented in several articles and books, maintenance of software accounts for between forty to eighty percent of the cost of a software package throughout its useful life [Ara85], [Kis83], [Mar83]. In 1976 it was reported that sixty to seventy percent of the Department of Defense's software dollar was spent on maintaining current software systems. In 1980 that figure rose to eighty percent. Another study showed that sixty-seven percent of data processing companies' costs were directly associated with software maintenance [Mar83].

With the rising cost of software maintenance comes increasing pressure on maintenance personnel and upon management dealing with maintenance in most companies [Bra85], [Col83], [Mar83]. In some cases, as more time is spent on doing maintenance of software, less time is spent in the development of new software. This trend has evolved to the point where some companies have gone bankrupt because they could no longer keep up with their new software development obligations [Mar83].

The problems mentioned above show the critical need for research in the area of software maintenance. The question of where to concentrate research dealing with software maintenance remains unanswered. In order to understand where research needs to be concentrated, it is necessary to look at the problem that is central to all maintenance tasks. This problem deals with the maintainer's need to learn what a program does and how it does it. The time needed to learn about a program is one of the major stumbling blocks associated with software maintenance [Mar83], [Par82].

Categories of Maintenance

To understand how to best solve the problem dealing with learning about a program, it was necessary to learn more about maintenance of software. There are many different categories of software maintenance. MAT1 is designed to be useful while doing any type of maintenance task. The tasks involved with software maintenance have several different classifications.

Most articles dealing with the topic of software maintenance divide maintenance into the three categories defined by E. B. Swanson [Swa76]. In his article Swanson divides software maintenance into the categories named Corrective Maintenance, Adaptive Maintenance, and Perfective Maintenance. Corrective maintenance is performed to identify and correct software failures, performance failures, and implementation failures. Adaptive maintenance is performed to adapt software to changes in the data or processing environments. Perfective maintenance is performed to enhance performance of the software, improve cost-effectiveness, improve processing efficiency, or to improve maintainability.

Another categorization of maintenance tasks was proposed by John Reutter [Reu81]. Reutter divided maintenance tasks into emergency repairs, corrective coding, upgrades, changes in conditions, growth, enhancements, and support. Emergency repairs are performed when immediate repair of a program is necessary to continue service to users. Corrective coding is performed to utilize system resources or to meet the design specifications. Upgrades are performed to adapt to the changing regulations in the business world. Growth is performed to adapt to changes in data or in the addition of new users or programs to the system. Enhancements are performed in response to user requests for updates to the current system. Finally, support is performed to explain system capabilities and to measure system performance.

Changing Attitudes Towards Maintenance

More recent publications show that the attitudes towards doing maintenance tasks are changing. These views are expressed in the calls for more extensive research, development of tools, and the expressed need for experts in the maintenance field. Many of the papers written in the IEEE Conference on Software Maintenance that was held in 1985 express a changing attitude towards software maintenance [Bra85], [Let85], [Wed85]. People that can do maintenance tasks efficiently are becoming more important to large corporations and to smaller businesses that use computers, as well.

The 1985 IEEE Conference on Software Maintenance breaks the study of maintenance into several areas of concentration [IEE85]. These areas include production of maintainable software through better design, development of tools for aiding maintainers with software maintenance tasks, development of measures and testing techniques for maintenance, and approaching software maintenance from a managerial point of view. All of these different categorizations offer many opportunities for study.

Development of Maintainable Software

From the material that was available, the main thrust in dealing with the problem of software maintenance is centered in the area of developing maintainable software [Hec83], [Mar83], [Mil83], [Rom85]. One of the topics stressed is the use of structured programming techniques and modular code when writing the original program. The use of standardized code will help in maintenance since the code will be known by the maintainers and can be easily accessed for changes. Some methods associated with standardized code include having all program code written by one person, programming functions and storing them in a library of code so that new programs can be constructed using the different functions, and using automated code generators to do as much of the programming as possible [Abb83]. The use of old code that has been developed and maintained, for new program development is stressed in some articles [Lan83]. By reducing human involvement in the code development process, while using automatic code generators, better and more maintainable code can be produced [Mil83].

A critical area dealing with maintainable programs is the need for the design phase to address maintainability [Mar83]. By designing modular programs, many of the problems of adding and perfecting the code can be ameliorated. Programs that are not written using modular programming techniques can not be as easily maintained as can programs that are written using structured programming techniques. The modularity of a program should be developed during the design phase of the software life cycle.

Introduction of good programming methodologies during design will eventually aid the maintainer in the maintenance phase.

Documentation

Another factor that is important to the production of maintainable software is the need for good program documentation. When good documentation accompanies a program, the task of maintaining that program becomes less cumbersome. When the original programmer is not available to maintain a program, documentation becomes a vital source of information to the maintainer. The maintainer must learn the function of the program in order to be able to perform the maintenance task. The development of automatic documentation generators along with automatic code generators is an idea that many authors have proposed that will help to solve the problem involving documenting a program [Col83], [Fay85], [Hou83], [Kuh85], [Wal85].

The fact that documentation is one of the main sources of information about a program that is to be maintained is significant enough that I incorporated a documentation search and display feature into the tool that I developed. Documentation is a good source of learning material and, because my tool is aimed at cutting down on the time that necessary to learn about a program that is to be maintained, this feature was added to the tool.

Measures of Maintainability

Another topic that is closely associated with maintainable software is the development of measures of maintainability of program code. Measures of maintainability include understandability of code, reliability of code, code testability, code modifiability, code portability, efficiency of code, and useability of program code [Mar83]. Code that is easily understood, reliable, easy to modify, efficient, and can be easily tested is much easier to maintain than is code that does not fit into these categories [Kis83]. Several studies on these categories show a significant reduction in time that is needed for maintenance personnel to do a maintenance task on code that fit into the categories that are mentioned above. Code that does not fit into the categories discussed previously do not score well in measures of maintainability.

Testability of code is a measure of maintainability that has already been studied in depth. Much work has been done in the testing phase of the software life cycle, and recent studies link the testability of code to maintainability of code [Mar83]. Many methodologies for testing code have already been developed. If a program can easily be tested, it is evident that maintenance work is reduced since testing of modified code is one of the tasks associated with software maintenance [Chr83], [Cur83], [Wal85]. Often, easily tested code is more modular in form, which is also an area that is being stressed by maintenance experts.

The need for software quality assurance as part of a maintenance effort is stressed in several articles [Bow83] [Day85]. Many times changes are made with the idea that they will correct mistakes, when in fact they cause many new problems [Bow83]. Software quality assurance deals with the fact that a program does what it was designed to do. It also deals with the correctness of the results produced by a program. Software quality is linked to maintenance of software as well. Since maintenance involves introduction of new code into a program, this code must still produce the results that the programmer desires. Quality assurance is therefore necessary in maintenance tasks as well [Day85].

Management of Maintenance

Management personnel often view maintenance of software in a very different perspective than maintenance personnel. Managers have to take into account the costs and time involved in doing maintenance tasks. They must also allocate time for new software development. Many times, managers face the question of whether to maintain a current program or to allocate time for development of new software [Mar83].

Management personnel must also decide how maintenance fits into overall software management. When a new system is installed, the addition of more users could cause a need for more maintenance. Another problem that can arise is the cost of purchasing new hardware or software when the current system no longer meets the needs of a company. All of these problems and many more must be solved by management when they are making decisions dealing with maintenance [Bra85], [Col83], [Dea83].

One proposal to solve some of the problems that management has is to introduce a better path of communication between maintenance personnel and management. The idea of using a standardized maintenance request form that will show the need for maintenance of software is one idea that was presented [Par82]. Other ideas are presented in the 1983 IEEE software maintenance workshop and in the 1985 IEEE conference on software maintenance. These ideas include better testing of code and better quality assurance techniques. Each idea stressed the need for better communication between management and maintenance personnel.

Maintenance Tools

The area of research dealing with tools to aid in software maintenance is relatively new. The many problems associated with software maintenance show the need for the development of tools to aid in the maintenance phase. Many tools have been developed to produce code and documentation [Bus85], [Har83], [Raw83]. This type of tool does not aid persons doing maintenance tasks, though. The question of what other tools need to be developed still remains.

The learning phase of maintenance is one area that has not been targeted for tool development. This phase is important to all maintenance tasks. Maintainers must take the time to learn the function of a program to be able to make the necessary changes to it. The question of what are the best sources of information for use in learning about a program needs to be examined. The answers to this question can be used to learn more about the learning associated with software maintenance.

Sources of Maintenance Information

Documentation is one of the sources of information that can help maintenance personnel to learn about the code that they are to maintain. The best type of documentation display is an on line system that is available at the maintainers request [Fay85], [Kuh85], [Raw83].

Another source of information that is useful comes from the history of past changes to the program. A history feature in a tool that helps with maintenance tasks is vital to aid in the learning process associated with maintenance of programs [Par82]. By having a history feature much of the redundancy in the learning process for maintaining programs can be eliminated. The maintainer will not have to retrace all the steps that was necessary to make a previous change when a history of changes is available for review. Also previous changes will help to identify links in the program that can cause hidden side effects when a change is made. A history of changes will help to reduce the time necessary to make a new update to a program. A knowledge of a history of changes is also useful in determining which changes were successful and which were not [Mar83].

I have included a history feature in MAT1 that will search for changes made to a module or variable. Over time rules of thumb can be developed or recognized that produce correct results when maintenance tasks are performed. By following successful methodologies the rules of thumb that an expert uses to make successful changes can easily be learned by new maintenance personnel. For example, when a particular section of code is changed and tested, the links it has with the remaining code should be identified. A maintainer can enter this information into the knowledge base so that the tool can inform future maintainers of these links and will therefore save time in correcting hidden errors that may not have been clearly identified. MAT1 is designed to reduce the time necessary for learning about the program that is to be maintained.

Other Important Maintenance Topics

Many other parts of a program are also important to doing maintenance tasks. The information that can be learned by having these program parts available for review is a vital part of MAT1. Variables, modules, parameters, and sections of code are important items in learning what the program does and also learning about what effects a change can produce elsewhere in the code [Col85]. By searching for variables used in a program, the maintainer can see what effects a change he makes might cause elsewhere in a program. This is also true when studying parameters that a routine uses.

Searching for keywords and sections of code will also help a maintainer to learn about the programming techniques that were used by the original programmer.

Programming style is closely associated to the points introduced in the articles on designing maintainable software [Par82]. When a programmer uses a structured programming approach module, names and parameters of modules are important along with local variables. These parameters may cause hidden effects to a program when they are changed. Local variables may be aliases for other variables in the program and also could cause problems if changed. A search for these different items, their purpose, and date of use and aliases can all provide vital information to a person doing a maintenance task. Having the ability to quickly search for a module name, parameter, or variable can speed the testing as well as the change process. In the testing process, hidden side effects can be easily determined and compensated for. A search also allows the program to worry about code location and eliminates one more concern for maintenance personnel. MAT1 has a search facility built into it that displays information graphically. This information can be used to obtain other information about the program.

If a programmer did not use a structured programming approach, then one must have the ability to search for just variables and their aliases. This style of programming has been shown to take much more time to learn than does a structured approach [Mar83], [Par82].

Maintenance Experts

The fact that maintenance is often performed by novice programmers can cause many problems. Often novice programmers spend much more time in the learning phase of maintenance than do experts. One solution to the maintenance problem is to have experts doing maintenance tasks.

Maintenance personnel that are able to develop successful techniques for solving maintenance problems, are very useful to their companies. As maintenance personnel become experts in the field of maintenance, they will be able to form methodologies for doing maintenance tasks. These methodologies are often 'rules of thumb' that are developed over time. As rules of thumb or heuristic knowledge is developed in the maintenance phase, expert systems can be developed that will help to automate the tasks dealing with maintenance.

Artificial Intelligence & Expert Systems

The second major area deals with artificial intelligence programming and problem solving techniques. The use of a knowledge base that will allow the tool to acquire knowledge about a task is one of the current subjects being studied by artificial intelligence practitioners [Dym84], [Ste84]. Several programming environments suitable for expert system development are available. I looked at Ops5, Personal Consultant Plus, and LOOPS.

The use of artificial intelligence ideas and programming practices open many new possibilities for development of tools to help in programming tasks. With the new knowledge engineering techniques available, expert systems can be developed that can help with most any problem.

An expert system, as defined by Clive L. Dym from the Xerox Palo Alto Research Center [Dym84], "is a computer program that performs a task normally done by an expert or consultant, and in so doing it uses captured heuristic knowledge". He describes heuristic knowledge as rules of thumb or techniques that an expert develops over the years that are easiest for problem solution. This is the type of program that would cut time in maintenance of software and would also be usable by both expert and novice maintenance personnel since many decisions could be made by the expert system based on heuristic knowledge that it has gathered.

LOOPS

LOOPS was chosen as the programming environment to be used for developing the expert system. LOOPS incorporates four programming paradigms. These paradigms include traditional procedural-oriented programming, object-oriented programming, data-oriented programming, and rule-oriented programming. LOOPS allows a programmer to develop software that uses one or all of these paradigms.

To use LOOPS one must first learn Lisp which is used as its procedural-oriented paradigm. Object-oriented programming is based on the idea of data abstraction. Objects have aspects of both procedures and of data. Different "procedures" are invoked by a message passing scheme designed by the programmer. These two paradigms, were used for the development of MAT1.

The data-oriented approach to programming in LOOPS allows the programmer to invoke independent processes based on actions on data. This concept allows the reading or writing of data to invoke processes as well as updating the data value itself. The last paradigm is rule-oriented programming. In this paradigm cause-effect rules are developed to control program action.

Several articles by members of the Xerox PARC group explain the uses of object-oriented programming techniques in the development of expert systems [Ste84] [Dym84]. These articles show how to use the object-oriented paradigm to pass inheritance traits from higher level objects to lower ones. The message passing techniques invoke methods (procedures) when messages are passed.

Object-oriented programming also allows the programmer to work in an environment in which data structures for holding and processing data do not have to be any concern of the programmer. This allows programmers to develop programs that are not dependent upon data structures and so allows a much more flexible approach to programming.

Two useful features of LOOPS are the mouse and menus [Ste85]. When developing objects and relationships between the objects in the class hierarchy, one does not have to worry about the code that is needed to create a new class or to determine the links to other objects. The LOOPS environment produces much of its own code in most circumstances. Only when the user needs to develop a local procedure (method) does any writing of code take place. This frees the programmer from much of the system dependent programming and allows new programs to be developed quickly [Bob83]. Other features that were important to the implementation of MAT1 included system input and output and the use of the Interlisp-D window and menu facilities.

My thesis implementation makes extensive use of the window feature to display the material that is necessary for completion of a maintenance task. The bit map feature is also used to aid in making windows easily identifiable to users. The windows allow menus that are mouse driven to perform the desired tasks.

Conclusions

Software maintenance has been recognized as the most expensive phase of the software life cycle. Since maintenance has not been studied in depth, it offers an interesting area of study. There are many problems that must be solved in order to reduce the costs and time necessary do to a maintenance task.

Several problems are associated with the maintenance phase. These include: 1) increasing maintenance costs, 2) assigning novice programmers to maintenance tasks, and 3) management and maintenance personnel viewing software maintenance from different perspectives. If the time to do a maintenance task can be reduced, the costs associated with maintenance can be reduced as well.

Recent study of maintenance has been centered in developing maintainable code, development of tools to aid in the maintenance phase, and studies of how management personnel can be helped with maintenance decisions. I feel that a major problem in maintenance of software is associated with the amount of time necessary to learn about a program that is to be maintained. In order to solve this problem, the development of an expert system using the LOOPS programming environment was explored and implemented. The development of a knowledge based tool will help to solve the software maintenance problems that maintainers are facing.

Chapter 2

Requirements

INTRODUCTION

Not all maintenance tasks require the same information to complete them. A tool that will aid in all maintenance categories will be most useful to maintenance personnel. The requirements or features that are general to all types of maintenance are important. Each different categorization of maintenance has unique requirements or the need for specific knowledge to solve tasks in that category.

A clarification must be made at this point. The information that MAT1 returns to a maintainer will be called a fact about a program. Any information returned from documentation or a history of changes associated with a program will also be considered a fact about the program. The inferences that a maintainer makes about the facts that the tool returns to him will be referred to as knowledge learned about the program.

Knowing what facts are needed to do maintenance of software is important. The sources of facts about a program become critical to the development of an expert system that must draw on a knowledge base to help to solve a problem. By learning what sources of facts a maintainer uses to complete a maintenance task, these sources can be incorporated into the expert system that is to aid the maintainer. After having done maintenance work for a period of time, maintainers will develop a set of rules that they will follow when solving maintenance problems in the future. This heuristic knowledge can be programmed into the expert system to allow it to help the maintainer to make decisions.

SOURCES OF FACTS

First, it is necessary to determine what facts are needed to do software maintenance. At this point the definition of software maintenance is useful in helping to determine what facts are needed to do a maintenance task. The definition of maintenance of software, as stated by James Martin and Carma McClure authors of the book "Software Maintenance, the Problem and its Solutions", [Mar83] is: "maintenance is any changes that have to be made to software after it has been delivered to a customer or user". Maintenance can be categorized, according to E. B. Swanson, [Swa76] into corrective maintenance, adaptive maintenance, and perfective maintenance. A person must use knowledge and facts that are specific to each type of maintenance in order to complete each unique type of maintenance task.

Corrective Maintenance

In the task of corrective maintenance, failures in a software package must be corrected so that the program will do what it was designed to do. The facts that are needed to do corrective maintenance tasks include the failure or error in the program and also that portion of code that is causing the failure in the system. By having these facts available, a maintainer can easily find where the error occurred and where to make the necessary change. To gather the facts necessary to make a correction a tool must be able to search to for variables, modules, parameters, or sections of code. A tool that produces facts about the location of a specific item in the program code will aid the maintainer in acquiring the knowledge needed to find the location of the problem.

The need for corrective maintenance is usually caused by a semantic or logic error in the program code. Syntactic errors should be flagged by the compiler and so do not enter into software maintenance. Semantic errors are not as easily corrected. The maintainer must learn what the original programmer was trying to do logically during the development of the code. Program documentation could be used as a source for obtaining facts about the program that will help the maintainer to make a correction. The maintenance personnel must use their own knowledge to correct an error in the meaning or logic of the code. After having done many maintenance tasks, maintainers will develop rules of thumb that they can use to solve a problem A tool to aid in maintenance should be able to gather heuristic knowledge based on the techniques a maintainer used to solve the problem. By gathering this knowledge that is learned by the maintainer, the program will be able to produce more facts about the program on its own.

The problem of what facts are necessary, is further compounded when one realizes that each person has their own programming style or technique. Persons doing maintenance tasks must also acquire knowledge about the style or techniques used by the original author of the code in order to complete the job. By having a search feature available in a tool to aid in maintenance, the problems dealing with program style can also be addressed. The search feature will help to reduce the time necessary in learning about program style.

One method of obtaining knowledge of the style or techniques that the original programmer used is to have available the documentation for the software being maintained. Good documentation can provide facts that will help maintenance personnel to acquire knowledge about the function of a section of program code and also about the style of programming that the author used. Documentation can therefore be used to provide vital facts to an expert system being used to aid in the maintenance of software. This information could include items such as the purpose of the code, how it was written, when it was written, internal code documentation and so will be included in the tool being developed. If a maintainer can display documentation dealing with specific items such as purpose of code, date written, and notes on how and why the code was written much time can be saved by looking at facts that deal with the problem at hand. Extraneous information can be eliminated, and only important facts displayed. MAT1 has this capability built into it.

Another method of obtaining knowledge of the style or techniques that the original programmer used is to be able to trace through the code with the help of the tool. This trace should be able to find variable, module, and parameter names, and should display facts about each. The trace facility that MAT1 uses does not do a run-time search, but only searches for items in the actual code. This type of tracing will aid in locating variables and determining their use along with being able to identify them as being global or local variables. Tracing will also aid in developing a knowledge of the relationships between modules, as well.

Adaptive Maintenance

The second type of maintenance described by Swanson is that of adaptive maintenance. When an environment changes, a package often must be updated in order to adapt to the new environment. This adaptation can include many different types of changes to a software package and so many different types of knowledge must be incorporated when doing this type of maintenance work.

For example, if a database management system were to be implemented to replace an existing part of a system, this would require knowledge of not only the database management system, but also of the system into which it was being added. Some of the facts that are needed for this type of maintenance include: 1) a knowledge of the data structures to be used to store the data, 2) the hardware requirements necessary to implement the new system, 3) a knowledge of the existing system and many more too numerous to mention. This provides for a very complex maintenance task. An easier task from the point of maintenance work would be that of adaptation of a software package to a new type of hardware. The knowledge that would be required to complete a task of this type would include how the software interfaced with the hardware. This interfacing code would be the only portion of the code that would need to be modified if the code was modular in design. A knowledge of the hardware is therefore necessary along with how the program accesses it. This knowledge though, is not incorporated into MAT1. MAT1 is not designed to display facts about hardware. The second example is less encompassing than that of adding a database management system to replace a portion of a software system, but it still requires an extensive knowledge of the hardware as well as the software of a system.

Most of the knowledge required to do these types of tasks is beyond the scope of this tool. Facts about the hardware as well as the software of a system would need to be available for a maintainer to do an adaptive maintenance task. Knowledge necessary for doing adaptive maintenance tasks varies widely and would incorporate the need for a very extensive knowledge base to store the facts needed to do this type of maintenance. The incorporation of such a large knowledge base into an expert system could become very cumbersome and therefore does not lend itself to this project, currently. This however does not mean that it could not be added in the future.

Even though the task of adaptive maintenance is very extensive in the amount of knowledge necessary to complete it satisfactorily, the basic functions of this tool would be very helpful. The features that this tool has will allow it to be useful when working on an adaptive maintenance task. Being able to display and search for facts about a large system would help to reduce the time necessary to complete the learning phase of an adaptive task.

Perfective Maintenance

The last and most significant type of maintenance task is that of enhancement or perfective maintenance. This type of maintenance is the most used of any maintenance activity according to McClure and Martin [Mar83]. Most software is developed under strict time limitations, so not all software is as efficient as it can be. By efficient I mean that the program may not process data in the most efficient manner, or the program may not work in a cost effective manner. In order to compensate for this lack of efficiency the maintainers at some point must modify the program. The maintainers must have a working knowledge of what the program does and how it does it. McClure and Martin show that perfective maintenance can consume up to sixty percent of all the time spent doing maintenance tasks in many companies [Mar83].

In order to do maintenance of this type, the user must find the bottleneck in processing and examine what they feel can be done to improve the efficiency of the current system. Some facts that could identify bottlenecks could include time to run a module, or total lines of code. This information is currently not available in MAT1. Given this information, maintenance personnel must develop new code and incorporate it into the existing code in the correct position to produce the desired result which would be more efficient processing of data. A tool that can identify where different modules, variables, or pieces of code are located in a program would be very useful in doing this type of maintenance. Having these facts built into an expert system would greatly cut the time necessary to find particular section of code that needs to be changed or locations for insertion of new code, as the maintenance personnel would not have to keep track of the information on their own.

HISTORY OF CHANGES

Other knowledge that would be useful for an expert system to aid in the task of software maintenance would be a history of past changes. From the facts produced by a history feature, maintenance personnel can acquire knowledge on what has been done in the past to a particular program. This knowledge can be used to speed the change being made or as heuristic knowledge on how past changes have been made. For example, if a variable was changed in the past, links dealing with that variable in other parts of the program could be identified. Testing of the original change should identify problems and could be entered into a history of changes. This would help to eliminate hidden side effects when another change was made to the same variable in the future.

A history feature could also be used to identify problems with parameters in modules. The links between modules could be identified when a change was made. Having this knowledge available would help a maintainer to avoid future problems with changing values of parameters. A history of changes could also be used as a measure for doing rewrites to a section of code. For example, if some code section has been changed a certain number of times then it could be defined as a module or section of code that was in need of being rewritten.

A history feature incorporated into a tool to aid in software maintenance will help maintainers to acquire knowledge from past changes. This knowledge will help to eliminate duplication of methods that have been used to make changes in the past. Elimination of relearning what has already been done in the past will make it easier to implement new changes. In turn, this will cut the time necessary to learn about and to complete a maintenance task.

PROGRAMMER'S VIEW OF A PROGRAM

Another important area that must be looked at when one is doing maintenance is that of what was the original programmer's view of a program. The reason that this is important is that programming views can affect the amount of time that is needed to learn about how the program is performing its functions. Did the programmer view the program as a series of modules each performing a specific task, or did the programmer use a 'spaghetti' coding technique? This view of a program could also provide facts that could be used in an expert system being designed to aid in maintenance tasks. Newer techniques of modular programming verses older 'spaghetti' coding approaches would provide information on how to use a tool in either situation. If a program was written using modular coding practices then module names would be important, along with the parameters or arguments being passed to and from the module. The concept of global and local variables would also be useful in gaining knowledge to do a maintenance task when the program was written in a modular form.

On the other hand, if a non-modular programming approach was used to write the original program, then variables become more important in locating problems in the code and in making corrections or additions to the program. The data flow of the program would also be very important when non-modular programming approaches are used.

Conclusions

Once the general information about what needs to be fixed, added, or adapted has been determined for the specific program, the problem of how to best approach the maintenance task becomes important. Expert systems use heuristic knowledge or rules of thumb to help the user to complete the task at hand. For example, when a person does a task many times, they usually find some method that they will use again and again to produce the correct results in the easiest manner. Once this methodology has been identified it can be programmed into the expert system so that the system be able to draw from this information to aid the user. In order to gain heuristic knowledge, the tool must use some kind of technique that will gather the needed facts. This must be built in or used in the environment that will be used to develop the expert system.

To be able to assist with maintenance tasks, the tool must be able to be used many times to maintain the same program. From each of these uses, the tool will acquire heuristic knowledge on how to best solve each change that it is used for. MAT1 relies on the maintainer to enter facts about the history of changes and documentation in order to store the knowledge gained by the maintainer in a knowledge base which is actually a file storage system. These facts will be valuable when making future changes. The history function of the tool should be able to display facts necessary to aid in this function. If no history of changes is available the maintainer must be able to enter knowledge into the tool that will help to make the decisions necessary for the specific application in the future.

Each category of maintenance has information that is specific to that type of maintenance task. A tool that can be used to aid a maintainer with any type of task will be most useful to them. The different categories of maintenance have one major source of facts. This source is the documentation available for the program. Another source of facts is a past history of the changes that were made to the program. The programmer's view of a program and code writing style are also important sources of facts to a maintainer. Each of these sources of facts can be used to speed the learning process that is necessary in maintenance of software. Maintainers must learn the function of a program before they can modify it. MAT1 was developed with this thought in mind.

Chapter 3

Necessary Capabilities, Design,

Environment, and Implementation

INTRODUCTION

A tool that will aid in all categories of software maintenance needs to have several features. Each of these features should present facts about a program to the maintainer in a form that will be easy to understand and learn from. The goal of developing MAT1 is to cut the time needed to learn about a program's functions. The features that need to be incorporated into it include: 1) a documentation update and display, 2) a history of changes update and display, 3) an edit feature, and 4) a search feature that can be used to find specific items in the program code, documentation, and history of changes.

CAPABILITIES

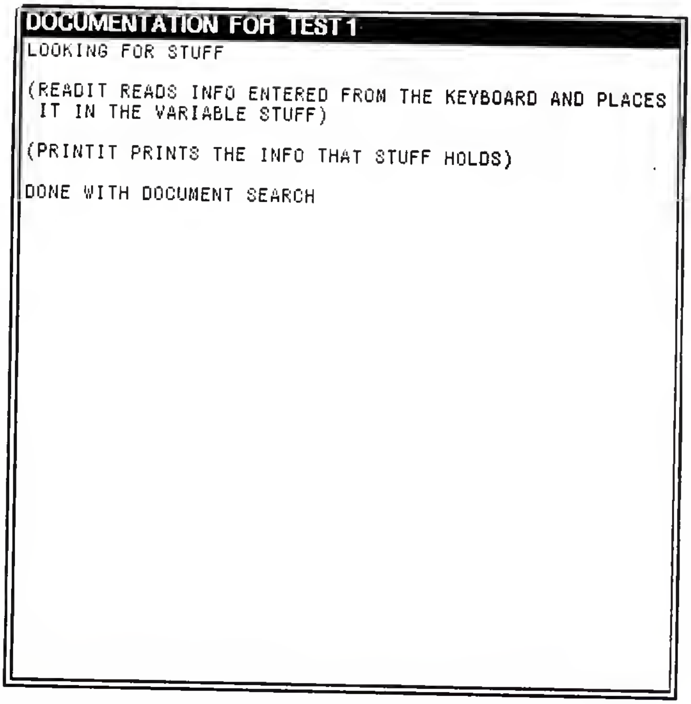
Documentation Update and Display

One of the needs of maintenance personnel that is general to all types of maintenance work is that of having an on-line documentation display and update feature. Documentation helps maintenance personnel to learn and understand how a program was written. This will also help maintainers to see why the program was written. A tool that is to aid in maintenance must have a documentation feature that can easily be viewed by maintenance personnel.

The documentation should be easily updated to accommodate changes that are being made to the program code currently. Even if no documentation is available upon the start of a maintenance task, the person doing the maintenance must be provided with the opportunity to document what they have learned about the program. This is necessary since the program may have to undergo changes in the future and the person who did the first update may not be available for consultation. If this is the case, new maintenance personnel must learn about the program for themselves. Documentation is one of the best sources of knowledge about a program when the original author is not available [Gi82]. This feature will reduce the time that is needed to learn about the functions and execution of a program by new maintenance personnel.

The Maintenance Assistance Tool (MAT1), will have access to on-line documentation for the program that is being maintained. The documentation is broken into several distinct parts. The first of these is notes on updates. MAT1 will be able to access different notes found throughout the program code or in a separate 'note' file. This will help to eliminate searching through manuals to find what a section of program code is doing. By keeping the documentation in small pieces, only information that is specific to the portion of code being changed will be displayed, eliminating the need for reading the whole manual or searching for a piece of documentation in an index.

MAT1 will also have the capabilities needed to access all on-line manuals for the program being maintained. This means that the maintainer can read manuals if necessary, although the option of only reading relevant information is available. Internal documentation other than notes is also available for display. A MAT1 example documentation display is shown in figure 1.



```
DOCUMENTATION FOR TEST1
LOOKING FOR STUFF

(READIT READS INFO ENTERED FROM THE KEYBOARD AND PLACES
 IT IN THE VARIABLE STUFF)

(PRINTIT PRINTS THE INFO THAT STUFF HOLDS)

DONE WITH DOCUMENT SEARCH
```

Figure 1: Documentation Display Window

If no documentation for a program is available, MAT1 will be useful in at least documenting the changes that are made. The tool will allow users to enter notes that will be incorporated with the programming code and also saved in a note file. This will provide information on what the maintainer learned and did when they worked with the specific program.

History of Changes Update and Display

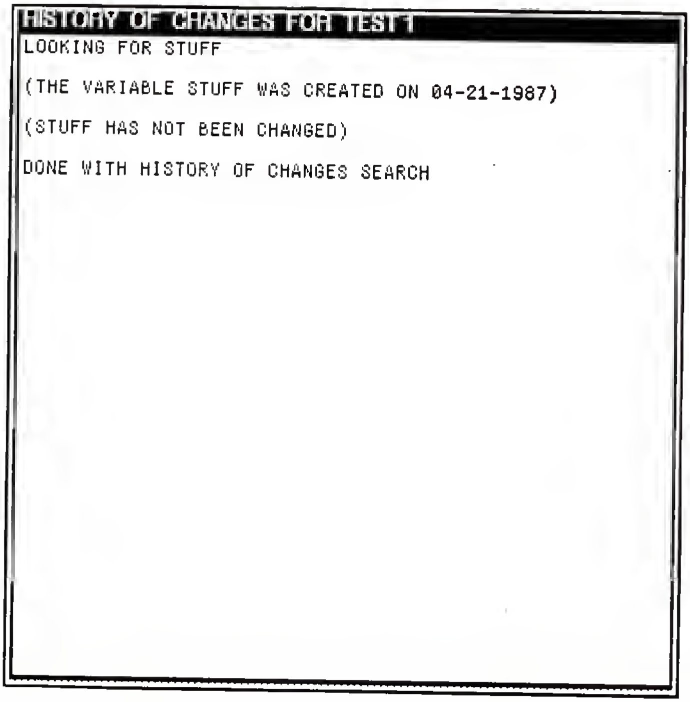
A feature that this tool has incorporated into it that is closely related to the documentation feature is that of a history of changes. By including a history feature in a tool to aid in software maintenance, persons maintaining the program can see what changes have been made in the past. From this, the location of changes can be learned, so that if someone is changing the same piece of code they can learn from past maintenance activities. The history feature can keep track of what effects resulted in the change being made. This will be invaluable for doing current changes as knowledge learned from making past changes will provide a basis for doing the current maintenance work. A history feature can help persons to make changes to the code faster, and with less unwanted side effects. By studying successful and unsuccessful changes that were made to a program, past mistakes in maintaining software can be avoided and correct decisions can hopefully be made more easily.

MAT1 is able to display a history based on the purpose of the change, the date of the change, or the type of change that was made. The purpose of a change can be used to learn more about what each module or variable is used for and what happened when they were changed. The date of change can help to show the order in which changes were made. From this maintainers can see the effects that changes in the past have made. This will aid them in learning the methods used to make the changes. By looking at the type of change, maintainers can gain a knowledge of what needs had to be met in the past, and from them they can determine if the changes they are making currently are relevant or will likely be successful. The history information will display facts on what happened when a change was made. If the change was successful and no side effects were recorded the maintainer can assume that his change to the same variable or module will also be successful and will produce no hidden side effects. Testing though, will be the only proof in this matter. A MAT1 example history display is shown in figure 2.

Search Feature

Another feature that is necessary is a search based on the programming approach that was used by the original author. When a modular approach to programming is used, a search for module names i.e. subroutine or function names, will be useful. Along with this, parameters the module uses should be identified to determine what changes to a particular module might affect other modules based on the fact that values in parameters are pass by reference. In other words the side effects that could be caused by changing a value in a module must be flagged and in some manner displayed as well. The hierarchy of the program can be learned by having a search and display feature as well. Closely related to the program hierarchy is that of data flow. Although MAT1 does not have a display for the data flow, a search and display feature to perform this function could be incorporated into the tool.

If the original programmer used a non-modular approach to programming the original code then some method of searching for variable names must be used. The user



```
HISTORY OF CHANGES FOR TEST1
LOOKING FOR STUFF

(THE VARIABLE STUFF WAS CREATED ON 04-21-1987)

(STUFF HAS NOT BEEN CHANGED)

DONE WITH HISTORY OF CHANGES SEARCH
```

Figure 2: History of Changes Window

will be able to query the tool as to the location of a variable or module name. If a program used line numbers, this could be easily displayed, as well. The search could then be used to aid in keeping track of where changes need to be made. This will free the user from having to remember all locations that must be changed. By incorporating this feature into the tool much time in paging through a program source file will be eliminated.

MAT1 is able to display modules when a modular programming approach was used to write the original program. In this display the purpose of the module, its parameters (input and output variables), and date last changed are shown. This will allow the maintainer to search for all modules using a specific variable or a specific purpose. The MAT1 module search display is shown in figure 3.

When a non-modular programming approach was used to design the original program, MAT1 is able to search for specific variable names, or code purpose (if documented). For variables MAT1 displays purpose, aliases, range of values, semantics, and the date they were last changed. By displaying this information the maintainer can learn what the tool knows about each item. This will help the maintainer to cut the time necessary to learn about a program. The MAT1 variable search display is shown in figure 4.

PROGRAM STRUCTURE AND DESIGN

The LOOPS programming environment is uniquely suited to this type of tool because it allows gathering of hueristic knowledge from the different instances of the classes. When an instance is created it can be stored into a knowledge base for use at a later time. The program itself actually stores the information that the maintainer

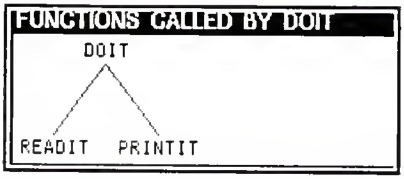


Figure 3: Module Name Search Graph

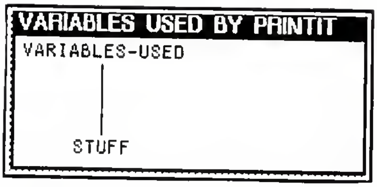


Figure 4: Variable Name Search Graph

enters in a file and then accesses that file to display facts about the program.

When using the object-oriented programming approach each instance of a class that is used in the program will inherit methods (procedures) from its supers or parents in the inheritance lattice for the program. Each of these methods can be invoked by using the "send super" message to a super of the current instance [Bob83]. By using this programming approach, the messages passed to a class instance can invoke new features without affecting the current code for the rest of the program.

Along with this approach to programming, the procedural approach is added to complete the necessary processing. Figure 5 shows the class inheritance lattice for the tool. This inheritance lattice shows the inheritance patterns for methods that are available in MAT1. The tool incorporates a documentation display and update feature, a history of change feature and a search based on whether a program was written in modular or non-modular form.

The maintenance object has a method that creates an instance of the documentation, history, or program objects based on what the maintainer wants to do currently. This method will be the main control for the creation of instances of the different classes that need to be used for the maintenance task. Once a selection is made, the user will be placed into a new window to do that portion of the maintenance work. A portion of the tool that is hidden from the user is the use of a knowledge base to store the information gathered by an instance creation. This will help to gain the heuristic knowledge that is necessary for the expert system to be able to aid in the decision making process.

Environment

Once the knowledge that is needed for incorporation into the system has been defined, and the program structure has been designed, the question of what programming environment best suits this type of tool development comes into play. The artificial intelligence programming environment used for this application is the LOOPS environment, developed by researchers at the Xerox Palo Alto Research Center. LOOPS is designed for the development of expert systems [Bob83].

IMPLEMENTATION

MAT1 is implemented on the Xerox 1186 workstation. The program is written using LOOPS' object-oriented programming and procedural-oriented programming techniques. MAT1 requires that the program to be maintained be loaded and executed so that all the different parts of the program can be searched and information about that part displayed.

MAT1 has nine different classes (Figure 5). The history of changes, and documentation classes each have three methods. One method is used to display the history or documentation window. The second method is used to search for and display the documentation or history of changes, and the third method is used to enter new information about history or documentation. The maintenance class has one method that display the initial user prompt window and calls invokes all subclasses as necessary. The edit class has one method that calls the changes made class which has one method. The changes made method allows the editing of the program. The program view class has one method which invokes the nonmodular or modular search methods in the classes of the same name. The modular search method invokes the variable search or modular search methods. The nonmodular search method invokes the variable search method.

The variable search method is found in the variable names class, and the module search method in a class called module names.

The different methods range in size from two to five pages. There are about fifty-five lines of code per page. LOOPS itself generates approximately ten pages of code in the program. The total number of program pages is about sixty-five. As I learned more about loops the size of the program decreased as my programming became more concise.

Chapter 4

MAT1's Capabilities and How MAT1 Answers the Hypothesis

INTRODUCTION

The development of MAT1 evolved as I learned more about the LOOPS programming environment. When the original foundation for the implementation of a tool to aid in software maintenance was conceived and put into a planning stage, the use of LOOPS as the environment in which it would be written was not discussed. The first environment that was looked at was Texas Instrument's Personal Consultant Plus. The software from Texas Instruments was never obtained, so when the department obtained the Xerox 1186 workstations, the use of LOOPS for writing the tool became possible.

Learning the LOOPS programming environment is very time consuming. As I spent more time working in the LOOPS environment, it became evident that it is ideal for developing expert systems for solving problems.

As my knowledge of LOOPS expanded so did the capabilities of MAT1. The original inheritance lattice was not like the current one. In LOOPS the inheritance lattice can be easily changed in order to accommodate changes in design. LOOPS does much of the code necessary for keeping the inheritance lattice in order and so a programmer does not have to worry about changing code when a change is made to the lattice. This fact allows easy prototyping of programs which can be modified at a later date.

CAPABILITIES OF MAT1

The original design of MAT1 called for a documentation display and update feature. A history update feature and display was also needed. A search feature was needed to provide information about a program that is to be maintained using MAT1. Finally, some method of editing the program that was to be maintained was needed, too. Each of these capabilities was included in the final implementation of MAT1.

In addition to the documentation and history display features a search for specific information about each was added. In order to add this feature to the tool, it was necessary to store documentation and history text as a list in a file. This list can be searched for only specific parts (sublists). These sublists are then displayed showing only information about material that was requested.

All features of MAT1 are mouse driven. LOOPS and Interlisp-D provide for easy use of the mouse in selecting items from menus. The mouse can also be used to select nodes from graphs to obtain more information from a graph. The windows can be easily controlled using the mouse. These capabilities include moving a window, closing it, shrinking it into an icon, taking a picture of it and printing the window image. All of these features are controlled by the right mouse button. The middle mouse button is used to display all menus while using MAT1. The middle and left mouse buttons can be used to select items from menus or from graphs displayed by MAT1.

Information displayed in graphic form seemed to be the most direct method for displaying information about module and variable names for a program being maintained using MAT1. The use of graphs also allows use of the mouse to select nodes in the graph to obtain more information about the words or figures in the nodes. The search feature of MAT1 uses the Masterscope facility of Interlisp-D to determine which items will be displayed in the graph.

The Masterscope facility of Interlisp-D is very useful in determining the functions that a program calls. Masterscope can also find all variables used by a specific function or by a program. When Masterscope is combined with the grapher facility, the information returned by Masterscope can be displayed in a tree or lattice form. The idea of displaying information was originally taken from this fact.

Interlisp-D provides for easy displaying of graphs using the layoutsexpr command. This command will take a list as its input and display the contents of a list as a graph. By using the program name as the root node and the answer from the Masterscope questions "who does XXX call" and "who does XXX use", where XXX is the program, name information about what functions a program calls or what variables are used can be easily displayed.

Once the display feature had been completed, attention was focused on the documentation and history searches. As was stated previously, this information is stored in files as a list containing other lists (sentences). The first thing that had to be accomplished was to store the information in the correct form. The CONS operation will attach new information to a list but it attaches it at the beginning. This meant that the information had to be reversed before new addition could be added to the documentation and history lists. A recursive function had to be written in order to perform the addition of new information to the documentation and history lists.

Once the history and documentation storing conventions had been developed, it was necessary to create a search routine that would search the information in the lists and display only the information requested. Again a recursive function was developed in order to accomplish this task. The function takes the CAR of the list passed to it and searches for the item in the list. If the item to be found is a member of the list the whole list is displayed in the proper display window.

Because the search looked for only specific items in the list, it was also necessary to develop a display that would print out all documentation and history information. The most direct method of displaying files is to use a Tedit window to display the file information. By using Tedit to display the file contents, the user can correct or add new information to the file using this window rather than the regular input routine. Tedit commands must first be learned by reading the manuals accompanying the 1186 workstation.

The editing feature was the next task that was addressed. The Tedit feature is ideal for editing text, but it does not work for program editing. It was therefore necessary to use the Dedit Interlisp-D editor to solve this problem. When Dedit is used on a program a pop up menu appears asking what definition of the file name that was passed to it should be edited. The test program that is used for demonstration purposes has two types of definitions, a file definition and function definitions. A user of MAT1 should select the FNS (functions) option of the pop up menu to Dedit the program code.

MAT1 also has the capability of storing its information on hard or floppy disk. This convention allows the user to save disk space when using MAT1. The program is also available on floppy or hard disk. Again this is for the reduction of use of disk space.

Thirty-seven files are associated with the MAT1 program. Some of these are the demonstration program, but most are program files. All of these files must be loaded into the system so that the program will work correctly. To ease the task of loading these files a tool initialization and initial startup routine was written. This information is contained in a file called Toolinit. This file must be loaded and then executed to load all MAT1 files and to start the execution of MAT1. A more detailed description of how to use MAT1 and its specific features is described in the user's and technical manuals found in the appendices.

MAT1 AND THE HYPOTHESIS

The fact that MAT1 uses a 'knowledge base' to store pieces of information about a program that is to be maintained helps to show that expert systems can be developed to aid with maintenance. These facts can then be displayed when requested by a maintainer in order to help them to make decisions that will solve the maintenance task. MAT1 provides the user with an environment that can be used and learned very quickly so as not to distract from the maintenance task at hand. Because MAT1 is mouse driven, it is very easy to use. It allows the user to search for specific information about a program based on whether the program was written using modular or non-modular programming styles. Documentation and history information display and update features provide vital information to the maintainer about what the program being maintained does and how the program performs its functions. The history feature allows a user to determine what past changes have been made and whether they were successful or not. By having this information available about the program being maintained, the maintainer can learn about a program very quickly and can perform the maintenance task easily and efficiently with less change of making errors. Thus, MAT1 implements the idea expressed in the hypothesis. MAT1 is an example of the kind of tools that will aid maintainers in learning about the program they are to maintain.

Chapter 5

Conclusions and Future Research

CONCLUSIONS

In conclusion, maintenance of software has long been overlooked. To avoid costly maintenance bottlenecks, new tools have to be developed to help with the maintenance phase. Tools that can access distributed documentation and notes about programs will help to cut down the time that is necessary to learn the functions of programs that maintainers are to change. MAT1 is a tool that can be used to help to speed the learning process that is necessary for maintenance by providing easy access to documentation, history, and searching for specific program information. Documentation and history are two major sources of knowledge available to maintainers. This tool helps maintainers to use this knowledge to aid in the decision making process that is necessary for completing maintenance work.

The LOOPS programming environment provides an excellent facility for developing tools like MAT1. The LOOPS environment allows for easy change of program design. This fact allows programmers to easily develop prototype programs and to expand them into working systems. The use of the different programming paradigms allows for very flexible approaches to programming and for designing programs. The fact that LOOPS is not well documented though, slows the learning necessary to use the environment.

The Interlisp-D programming language also allows for many powerful features which can easily be added to an expert system. The use of windows, menus, graphs, and the 1186 mouse help to simplify the use of the programs that are written. The windows and graphs allow for easy display of information on the screen. The list processing and file handling capabilities of Interlisp-D are critical to the use and development of MAT1.

FUTURE RESEARCH

Features that are not general to all maintenance tasks can easily be added to this tool when using the LOOPS programming environment. By using either the object-oriented or data-oriented programming approaches, independent processes can easily be added to the system without affecting the other parts of the tool that are already running. This means that new features can be added without affecting the operation of the current version of MAT1.

Currently, MAT1 is limited to being able to work with Lisp programs. In order to expand the use of MAT1 to other programming languages, an editor for that language would have to be incorporated into the tool along with new search facilities (a scanner) so that the information about the program could be displayed graphically.

The program portion of MAT1 could be developed using more LOOPS conventions. As the program is currently written, most of the code is written using Interlisp-D. The code could easily be changed to incorporate more of LOOPS's conventions and features. The windows and menus could be defined as classes as could the different searches. Also, message passing between methods could be modified. The introduction of class and more instance variables into the system would make the code more efficient.

MAT1 has the capability of working on itself. This could provide an interesting area of future study as well. Other features that could be added to MAT1 include a variable alias search, module parameter identification, program listing display, and a language specific key word search.

I think that the area of software maintenance needs to be explored in more detail. As costs and problems associated with maintenance continue to increase, the need for study and expertise in this area becomes ever more evident.

BIBLIOGRAPHY

[Abb83] Abbott, S. F.: "Standardized Code" IEEE Software Maintenance Workshop, Monterey, CA. 1983. p. 54.

[Ara85] Arango, G. Freeman, P., and Pidgeon, C.: "Maintenance and Porting of Software by Design Recovery" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 42-49.

[Ban83] Bannai, K., Suzuko, M., and Terano, T.: "An Introduction of Maintenance Support Facility" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 180-182.

[Bas83] Bassett, P.: "Maintenance vs. Development: Two Sides of the Same Coin" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 244-245.

[Ber84] Berns, G. M.: "Assessing Software Maintainability" Communications of the ACM January, 1984, Volume 27, No. 1, pp. 14-23.

[Boe83] Boehm, B.: "The Economics of Software Maintenance" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 9-37.

[Bow83] Bowen, J. B.: "Software Maintenance: An Error-Prone Activity" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 102-105.

[Bra85] Branch, M. A., Jackson, M. C., Laviolette, M. C., and Frankel, E.: "Software Maintenance Management" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 62-68.

[Bre85] Brehl, A. W.: "Upgradeability: A Measurement of Quality" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 227-230.

[Bro85] Brownston, L., Farrell, R., Kant, E., and Martin, N.: "Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming" Addison-

Wesley Publishing Company, Inc. Reading, Mass. 1985.

[Bry83] Bryan, W. L. and Seigel, S. G.: "Configuration Management of Software Testing" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp 61-63.

[Bus85] Bush, E.: "The Automatic Restructuring of Cobol" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 35-41.

[Cha85] Chapin, J., and Faidell, G.: "Predicting Software Customer Support" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 128-134.

[Cha83] Chapin, N.: "Attacking Why Maintenance is Costly" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 251-252.

[Chr83] Christens, J. M., Stofko, M. J., and Gonsol, S. J.: "Insights and Experience Gained from the Development of an Internal and Independent Verification and Validation Program" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 125-129.

[Col83] Collofello, J. S.: "A Conceptual Foundation for Measuring Software Maintainability" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 253-254.

[Col85] Collofello, J. S. and Blaylock J. W.: "Syntactic Information useful for Software Maintenance." National Computer Conference (1985), pp. 547-553.

[Coj83] Collofello, J. S., and Woodfield, S. N.: "A Proposed Software Maintenance Environment" IEEE Software Maintenance Workshop, Monterey CA. 1983. pp. 118-119.

[Com83] Colter M. A., and Couger J. D.: "Management and Employee Perceptions of the Maintenance Activity" IEEE Software Maintenance Workshop, Monterey, CA. 1983. p. 130.

[Con83] Consol, S. J., Stofko, M. J., and Christens, J. M.: "Software System Change Control" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp 255-257.

[Cur83] Curtis, C. A., and DeHaan W. R.: "RXVP80: The Verification and Validation System for Fortran" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 75-77.

[DaR83] Da Rocha, A. R.: "Towards Maintainability Through Specifications Quality" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 246-250.

[Day85] Day, R., and McVey, T.: "A Survey of Software Quality Assurance in the Department of Defense During Life-Cycle Software Support" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 79-85.

[Dea83] Dean, J. S., and McCune, B. P.: "An Informal Study of Software Maintenance Problems" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 137-141.

[Dem83] Dempsey, J.: "The Design, Development, and Maintenance System" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 258-260.

[Dun84] Dunn, R. H.: "Maintenance and Modification" Software Defect Removal, McGraw-Hill New York, NY. May, 1984, pp. 304-323.

[Dym84] Dym, C. L.: "Expert Systems: New Approaches to Computer-Aided Engineering" Xerox Palo Alto Research Center, Palo Alto, California. 1984.

[Emb83] Embry, J. D., and Keenan, J.: "Organizational Approaches Used to Improve the Quality of a Complex Software Product" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp 131-133.

[Fay85] Fay, S. D., and Holmes, D. G.: "Help! I Have to Update and Undocumented Program" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 194-202.

[Foe86] Foehse, M.: "Introduction to Xerox 1186 A1 Workstation" Kansas State University 1986.

[For83] Forss, A. K., and Lundin, R.: "An Approach to Long Term Maintenance of ATE Software" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 173-179.

[Gla85] Glagowski, T. G.: "Using a Relational Query Language as a Software Maintenance Tool" IEEE Conference on Software Maintenance, Washington, D.C. 1985, pp. 211-219.

[Gus85] Gustafson, D. A., Melton, A., and Hsieh, C. S.: "An Analysis of Software Changes During Maintenance and Enhancement" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 92-95.

[Har83] Harada, J., and Sakashita, S.: "A Documentation Tool to Visualize Program Maintainability" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 275-280.

[Hap83] Hartmann, P. E., Feuling, C. L., and Hodil E. D.: "Tools in a Large Maintenance Environment" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 71-72.

[Hec83] Hecht, H.: "Software Requirements for the Maintenance Phase" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 145-146.

[Hin86] Hines, T.: "Loops: An Introductory Guide" Kansas State University, 1986.

[Hou83] Houtz, C. A., and Miller, K. A.: "Software Improvement Program - A Solution for Software Problems" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 120-124.

[How83] Howley, P. P.: "An Assessment of Software Testing Techniques for Maintenance" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 261-266.

[How85] Howley, P. P., and Reimer, G. W.: "Software Maintenance Criteria for Small Microprocessor-Based Systems" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 222-226.

[Kaz83] Kazlauski, F. A.: "Test Data Reduction Program: A Benchmark and Conversion Tool" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 73-74.

[Kis83] Kishimoto, Z.: "Testing Software Maintenance and Software Maintenance from the Testing Perspective" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 116-117.

[Kuh85] Kuhn, D. R., and Hollis C. G.: "Simple Tools to Automate Documentation" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 203-210.

[Lan83] Lanergan, R. G. and Grasso C. A.: "Reusable Designs and Code: A Strategy for Designing Software With Maintenance in Mind" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 55-56.

[Leh83] Lehman, M. M.: "Survey of Software Maintenance Issues" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp 226-243.

[Let85] Letovsky, S., and Soloway, E.: "Strategies for Documenting Delocalized Plans" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 144-151.

[Mar83] Martin, J., and McClure, C.: "Software Maintenance: The Problem and Its Solutions" Prentice-Hall, Inc. Englewood Cliffs, N.J. 1983.

[Mil83] Miller, J. C.: "Planning for Maintainability: Some Precautions" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 57-59.

[Min83] Minsky, N. H.: "Controlling the Evolution of Large Scale Software Systems" IEEE Conference on Software Maintenance, Monterey, CA. 1985. pp. 50-61.

[Mor83] Morgan, H. W.: "The Cost of Tools - Barriers to Technology Transfer in the Maintenance Environment" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 80-81.

[Mor79] Moriconi, M. S.: "A Designer/Verifier's Assistant" IEEE Transactions on Software Engineering, Vol. SE-5, No. 4, July 1979, pp. 387-401.

[Mye86] Myers, Ware: "Introduction to Expert Systems" IEEE Expert 1986, pp. 100-109.

[Par82] Parikh, Girish: "Techniques of Program and System Maintenance" Winthrop Publishers, Inc. Cambridge, Mass. 1982.

[Pee85] Peercy, D. E.: "A Framework for Risk Assessment of Software Supportability" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 120-127.

[Phi83] Philips, J. C.: "Creating a Baseline for an Undocumented System - Or What Do You Do With Someone Else's Code?" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 63-64.

[Pre83] Presser, L., and Hug, R.: "Change and Configuration Control Tool" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 271-274.

[Raw83] Rawlings, T. L.: "A Technological Approach to Automating Software Maintenance" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp 147-151.

[Rom85] Rombach, H. D.: "Impact of Software Structure on Maintenance" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 152-160.

[Rub83] Rubin, H. A.: "Macro and Micro-Estimation of Maintenance Effort: The ESTIMACS" IEEE Software Maintenance Workshop, Monterey, CA. 1983, pp. 78-79.

[Sch85] Schaefer, H.: "Metrics for Optional Maintenance Management" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 114-119.

[Sil83] Silverman, J., Giddings, N., and Beane, J.: "An Approach to Design-for-Maintenance" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 106-110.

[Smi83] Smith, G., and Von Schantz, C.: "Quality Assurance Plans for Software Maintenance" IEEE Software Maintenance Workshop, Monterey, CA. 1983. pp. 163-164.

[Ste83] Steifik, M., Bobrow, D. G., Mittal, S., and Conway, Lynn: "Knowledge Programming in LOOPS: Report on an Experimental Course" The AI Magazine fall 1983 pp. 3-13.

[Ste84] Stefik, M. and Bobrow D. G.: "Object-Oriented Programming: Themes and Variations" The AI Magazine (1984), pp. 40-62.

[Stm84] Steifik, M. J., Bobrow D.G., and Kahn K. M.: "Access Oriented Programming for a Multiparadigm Environment." Intelligent Systems Laboratory, Xerox PARC, Palo Alto, California. 1984.

[Swa83] Swanson, E. B.: "A Tutorial on Application Software Maintenance" IEEE Software Maintenance Workshop, Monterey, CA. 1983, pp. 2-8.

[Tay85] Taylor, B.: "A Database Approach to Configuration Management for Large Projects" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 15-23.

[Tou83] Touretzky, D. S.: "Lisp: A Gentle Introduction to Symbolic Computation" Harder and Row, New York, NY, 1983.

[Tur85] Turpin, W.: "Personal Consultant Plus: Expert System Development Tools" Texas Instruments, Inc. 1985.

[Wal85] Wallace, D. R.: "The Validation, Verification, and Testing of Software: An Enhancement to Software" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 69-78.

[Wed85] Wedo, J. D.: "Structured Program Analysis Applied to Software Maintenance" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 28-34.

[Wej85] Wedo, J. D.: "Systematic Problem Solving: The Link to Maintenance Solutions" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 231-235.

[Wig85] Wiggins, D. L.: "An Automated System for Controlling Operational Program and JCL Changes" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 2-5.

[Yue85] Yuen, C. K.: "An Empirical Approach to the Study of Errors in Large Software Under Maintenance" IEEE Conference on Software Maintenance, Washington, D.C. 1985. pp. 96-105.

Appendix A

USERS MANUAL

In order to be able to use the Maintenance Assistance Tool (MAT1), the user must be familiar with: the operation of the Xerox 1186 workstation, the use of the 1186 mouse, the editors Tedit and Dedit which are used in the Interlisp-D environment, and the hard and floppy disk drives of the 1186 workstation. All of this information can be found in the manuals provided with the 1186 workstation as well as in the Interlisp-D primer. If the user is not familiar with the items mentioned, one should first read the manuals provided, for an explanation of their use.

MAT1 is designed to aid maintenance personnel in obtaining the knowledge needed to perform a maintenance task. It provides an environment in which a maintainer can learn about the program that is to be maintained, from either documentation or history of changes for the program. The tool provides editing features that allow a maintainer to update a program as needed. When a change is made, new history and documentation can be entered to note the maintenance task that was performed. A search facility will find information about the different parts of a program and will display this information in graph form. Further use of the graph will display information about documentation and history for a selected item in the graph.

INITILIZATION

In order to start the Maintenance Assistance Tool, one must first boot the 1186. To do this, simply turn on the power and when the boot icons appear on the screen, press the F1 key to load the Interlisp-D environment. The loading of the Interlisp-D environment takes about two minutes, so be patient. While loading, the screen on the

1186 will be black. When loading is complete a new screen will appear with a prompt window, a tty window, a history icon, the LOOPS icon, and a logo window. The mouse cursor will appear as an arrow located in the upper left hand corner of the screen. When a blinking caret appears in the tty window, this shows that loading is complete and that the user can now use the Interlisp-D environment. The first thing that must be done is to set the time. LOOPS and Interlisp-D must have the time set before any file operations can occur. Since MAT1 uses LOOPS and many file operations, the time must be set for the tool to work properly. The command that is used to set the time is (SETTIME "MM-DD-YY HH:MM:SS"). Several different formats of the date and time can be entered, but the example will always work.

Once the time has been set it is necessary to determine of the file called GRAPHER.DCOM is available in the library of files on the system. The command that is used to check to see that the file is available on hard disk is (DIR '{DSK}<LISPFILES>LIBRARY>GRAPHER.DCOM). If the name of the file is returned the file is available on hard disk, but if the message File Not Found is returned, the file is not available in which case it is necessary to load this file from the floppy utility disks provided by KATO. MAT1 uses the grapher facility and so this file must be resident on hard disk in the directory mentioned above for the tool to operate correctly.

The user should now load the program that is to be maintained into the system. The program should then be executed so that all functions used by the program will be available in the environment. This is critical to the use of MAT1 as it relys on the fact that all functions of the program to be maintained are known in the tool environment. At present, only Interlisp-D programs can be maintained by the use of MAT1.

The next consideration that must be addressed is the one of whether MAT1 is resident on the hard disk drive of the 1186 or if it is on a floppy disk. If the tool is available on the hard disk drive the command (CNDIR '{DSK}<LISPFILES>AL>) must be entered. This command connects the user to the directory called <LISPFILES>AL> where the tool is resident. If the tool is not resident on hard disk the command (CNDIR '{FLOPPY}) must be used. This command will connect the user to the floppy disk drive directory and the tool can then be loaded from floppy disk.

The next step that needs to be taken is to load (MAT1) into the Interlisp-D environment. To do this a file called Toolinit must be loaded from the hard or floppy disk. The command (LOAD 'TOOLINIT) can be entered in either case as the system will know what file to load from the CNDIR command. Once the loading of TOOLINIT is complete, the user must enter the command (TOOLINIT). This command executes the commands necessary to load all the files that must be available for the use of MAT1. The TOOLINIT command will also start the execution of MAT1. The loading of the MAT1 files will take some time and it also requires the user to press the enter key when prompted in the tty window.

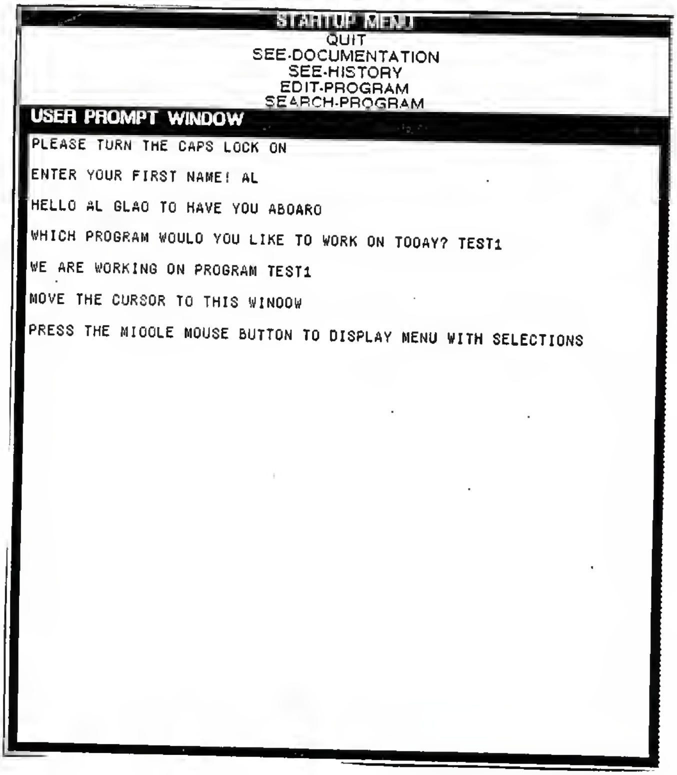
USING MAT1

When the MAT1 tool is started a logo window displaying Maintenance Assistance Tool will appear near the bottom of the screen. Another window will also appear in which the user is prompted to enter their first name and then the name of the file that holds the program that is to be maintained. This file should already be loaded and should have been executed, but if this is not the case, the user can load the program file and execute it when the blinking caret appears again in the tty window. The user is prompted to move the mouse cursor tothe window in which their name and the program to be maintained was entered. Next, the user should click the middle mouse button in order to display the menu of selections available with this window. The window is the User Prompt window. See figure A1.

Note:

When the program is first loaded it is necessary for the user to click the middle mouse button twice in each window in order to display the menus. This is necessary because all functions must be loaded into the system and then executed.

The maintainer can now move the mouse cursor to the menu that appears at the top of the user prompt window. In order to determine what each of the different items in the menu will do, press and hold either the left or the middle mouse button while the mouse cursor is pointing to one of the menu items. When the mouse button has been held for about three seconds a message will appear in the black Prompt Window explaining the function of each available menu item. See figure A2. If a maintainer does not want to perform a particular item, move the cursor to the next item but do not let up on the pressed mouse button. When the desired item that is to be executed is found, simply let up on the pressed mouse button. This action will select that item and will execute the selected function. As a user becomes familiar with the functions of MAT1, simply move the mouse cursor to the desired menu item and click wither the middle or left mouse button to select the desired function. When the user does not want to perform any item in the menu, move the mouse cursor out of the menu and then release the mouse button that was pressed. This explanation of the use of the mouse in the menu pertains to all menus that are displayed by MAT1.



```
PLEASE TURN THE CAPS LOCK ON
ENTER YOUR FIRST NAME! AL
HELLO AL GLAO TO HAVE YOU ABOARO
WHICH PROGRAM WOULO YOU LIKE TO WORK ON TOOAY? TEST1
WE ARE WORKING ON PROGRAM TEST1
MOVE THE CURSOR TO THIS WINOOW
PRESS THE MIOOLE MOUSE BUTTON TO DISPLAY MENU WITH SELECTIONS
```

Figure A1: User Prompt Window

Prompt Window -- Lisp: 22-Nov-85 Loops: 7-Jan-8

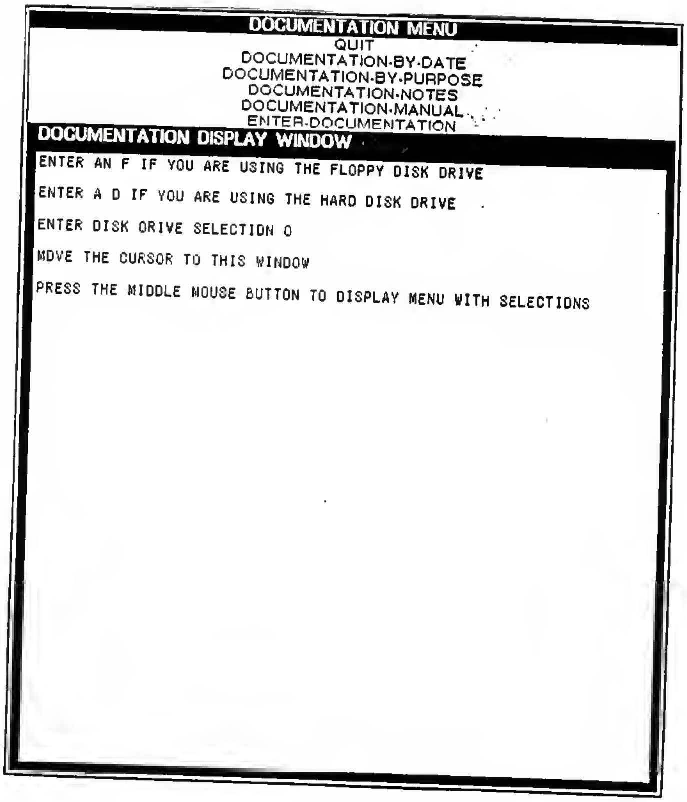
Figure A2: System Prompt Window

The menu items available in the user prompt or opening window include: Quit which is used to exit from MAT1 to the Interlisp-D environment, See-Documentation which opens the documentation window, See-History which opens the history of changes window, Edit-Program which opens the edit window, and Search-Program which opens the program view window. Each item that opens a new window will display a window that has another menu attached to it. The new menus can then be selected from to perform the desired function.

Documentation Search, Entry, and Display Feature

The See-Documentation selection opens the documentation display window. A user can then move the mouse cursor to this window and click the middle mouse button to display the documentation menu. See figure A3. The documentation display menu has the following items: Quit which closes the documentation window and redisplays the user prompt window, Documentation by Date which searches the documentation files for a specified date. The date must be entered in the form MM-YY-19YY. When this search is invoked a new documentation display window is created and the information about the current date is displayed.

The documentation menu also has a Documentation by Purpose item. When a user selects this item they can search the documentation files for a specific word that they are prompted to enter. Just as with the date search a new window is opened and all information about the word is displayed in the new window. A Documentation Note item works in the same manner as the purpose item except that the note is entered in the form NOTE#?, where ? specifies a number corresponding to the desired note to be displayed.



```
ENTER AN F IF YOU ARE USING THE FLOPPY DISK DRIVE

ENTER A D IF YOU ARE USING THE HARD DISK DRIVE

ENTER DISK DRIVE SELECTION D

MOVE THE CURSOR TO THIS WINDOW

PRESS THE MIDDLE MOUSE BUTTON TO DISPLAY MENU WITH SELECTIONS
```

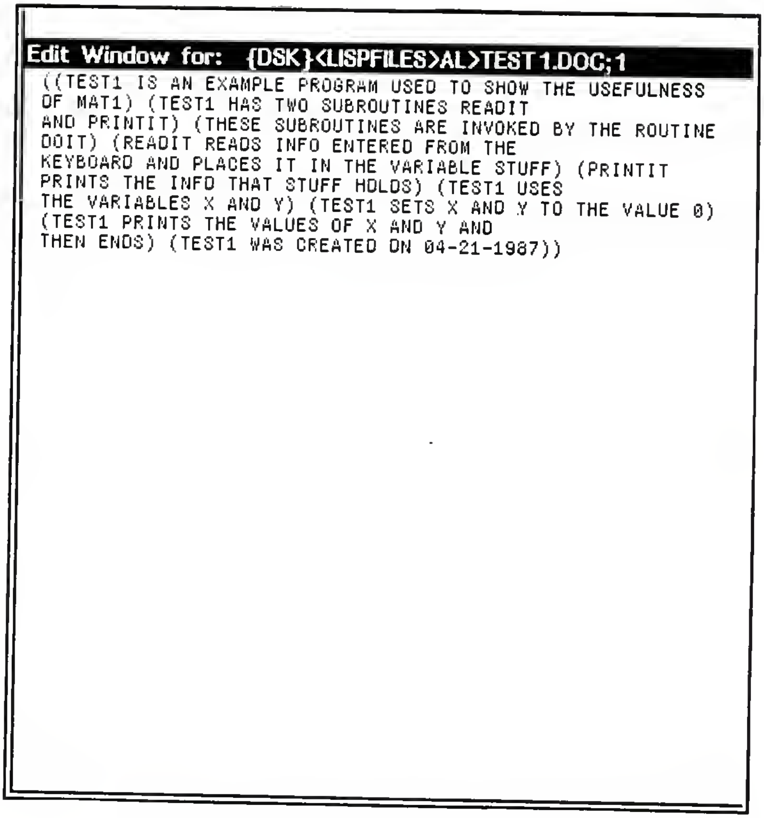
Figure A3: Documentation Window & Menu

A Documentation Manuals item is used to display all documentation available for the program that is being maintained. The documentation is displayed in a Tedit window in which the user can either just read the documentation or can edit the documentation if desired. See figure A4.

Finally, an Enter Documentation item is available in the documentation menu. This item allows the user to enter documentation for a program or for a change that was made to a program. The documentation is entered in the tty window. In order to stop the entry of a block of documentation or to stop a sentence, the user must press the space bar and then enter a period, question mark, or an exclamation point. When this is done the user is prompted if they want to enter more documentation or not. If N is entered the documentation is stored in a file that has the name of the program which was entered in the user prompt window with an extension of .DOC. If Y is entered the user can continue to enter documentation blocks or sentences as desired. When Enter Documentation is first selected the user is prompted to enter a Y if documentation files exist for the program being maintained for an N if no documentation files exist. The tool then takes appropriate action. If files exist the old documentation is displayed and new documentation can then be added to the existing file. If no documentation files exist, documentation is stored in a new file.

History of Changes Search, Entry, and Display Feature

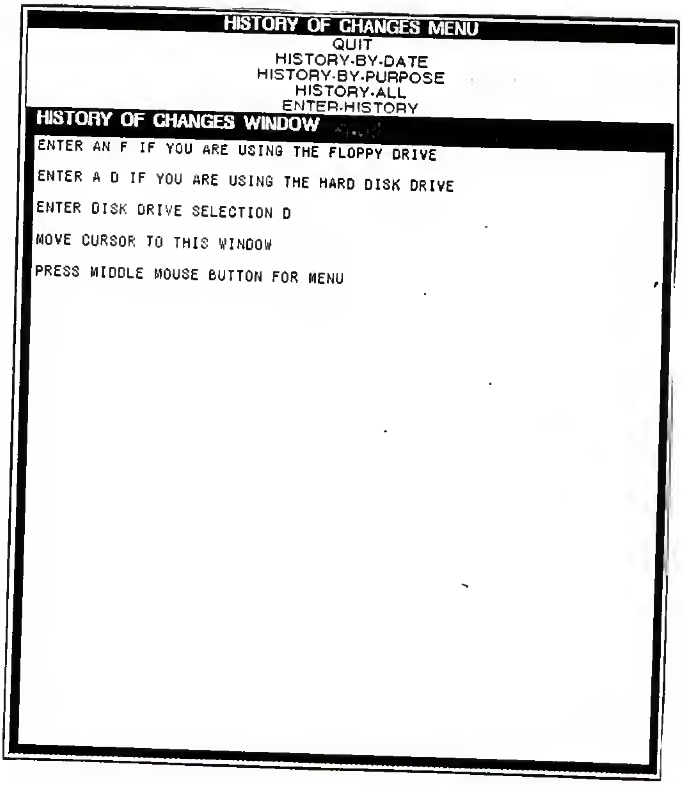
The See-History item of the user prompt window opens a history of changes window. The user can then move the mouse cursor to the history window and can click the middle mouse button for the history menu. See figure A5. The items in the history menu work on the same principle as do the items found in the documentation display



Edit Window for: {DSK}<LISPFILES>AL>TEST1.DOC;1

```
((TEST1 IS AN EXAMPLE PROGRAM USED TO SHOW THE USEFULNESS
OF MAT1) (TEST1 HAS TWO SUBROUTINES READIT
AND PRINTIT) (THESE SUBROUTINES ARE INVOKED BY THE ROUTINE
DOIT) (READIT READS INFO ENTERED FROM THE
KEYBOARD AND PLACES IT IN THE VARIABLE STUFF) (PRINTIT
PRINTS THE INFO THAT STUFF HOLDS) (TEST1 USES
THE VARIABLES X AND Y) (TEST1 SETS X AND Y TO THE VALUE 0)
(TEST1 PRINTS THE VALUES OF X AND Y AND
THEN ENDS) (TEST1 WAS CREATED ON 04-21-1987))
```

Figure A4: Document Manuals Tedit Window



```
ENTER AN F IF YOU ARE USING THE FLOPPY DRIVE

ENTER A D IF YOU ARE USING THE HARD DISK DRIVE

ENTER DISK DRIVE SELECTION D

MOVE CURSOR TO THIS WINDOW

PRESS MIDDLE MOUSE BUTTON FOR MENU
```

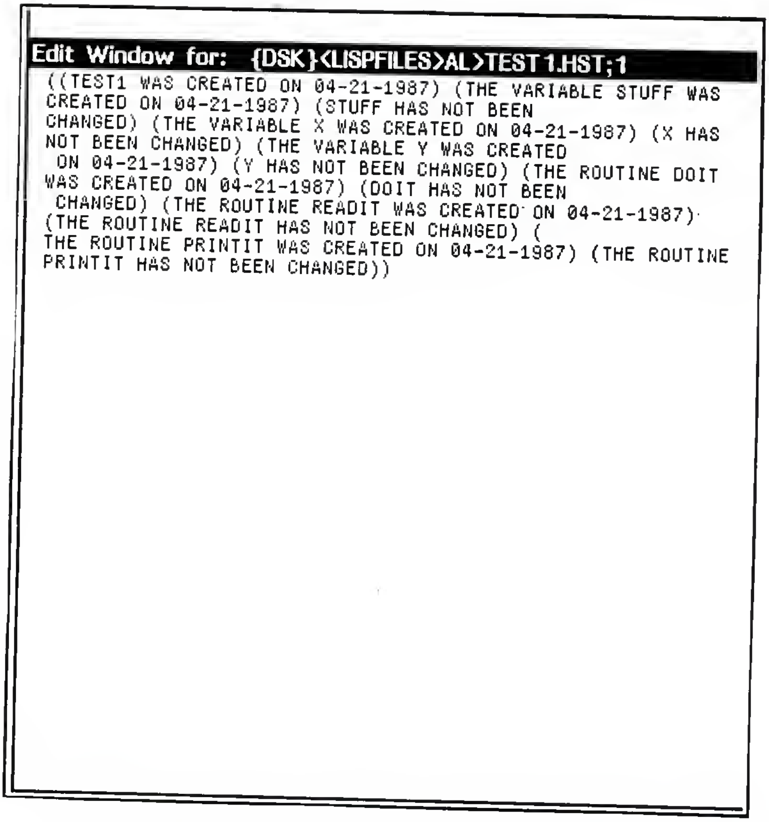
Figure A5: History Window & Menu

window mentioned above. The items in the history menu include: Quit which closes the history display window and redisplays the user prompt window, History by Date which works like the Documentation by Date function, History by Purpose which works like Documentation by Purpose, History All which works like Documentation Manuals, and Enter History which works like Enter Documentation except that history files have the extension of .HST. See figure A6.

When a search for documentation or for a history of changes is requested by selecting the proper menu item, the user is prompted to enter a D or an F. The entry of this letter determines which disk drive is searched for the documentation of history files. When a D is entered the hard disk is searched for the correct files, and when an F is entered the floppy disk is searched. This allows flexibility in the storage of history or documentation files.

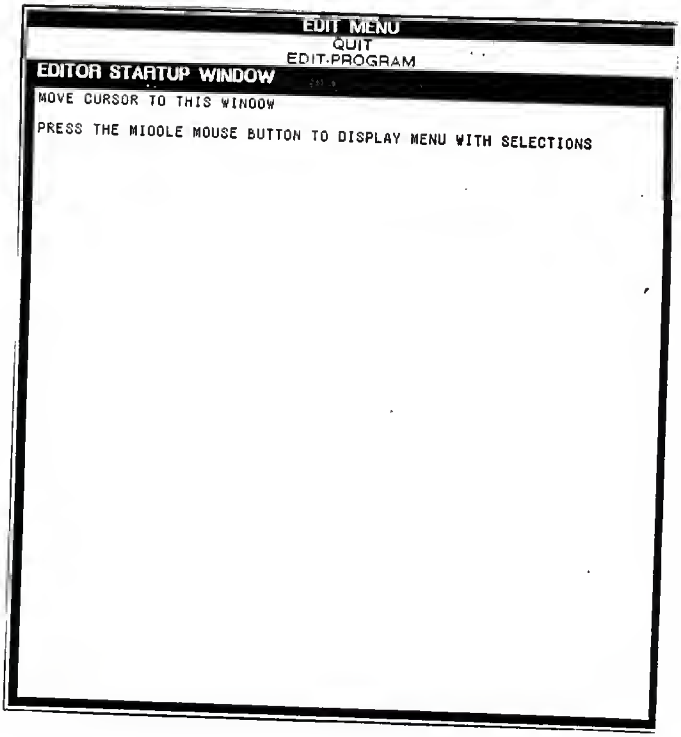
Editing Feature

The next item available in the user prompt window menu is Edit Program. When a user selects this item the edit program window is displayed. The user must then move the mouse cursor to the window and click the middle mouse button to display the edit menu. See figure A7. The edit menu has only two items available. These are: Quit which closes the edit window and redisplays the user prompt window, and Edit Program which allows the user to Dedit the program named in the user prompt window. A pop up menu will appear when the edit program item is selected. The user should then select FNS from the items available. This will cause a Dedit of the program functions in a Dedit window.


Edit Window for: {DSK}<LISPFILES>AL>TEST1.HST;1

```
((TEST1 WAS CREATED ON 04-21-1987) (THE VARIABLE STUFF WAS
CREATED ON 04-21-1987) (STUFF HAS NOT BEEN
CHANGED) (THE VARIABLE X WAS CREATED ON 04-21-1987) (X HAS
NOT BEEN CHANGED) (THE VARIABLE Y WAS CREATED
 ON 04-21-1987) (Y HAS NOT BEEN CHANGED) (THE ROUTINE DOIT
WAS CREATED ON 04-21-1987) (DOIT HAS NOT BEEN
 CHANGED) (THE ROUTINE READIT WAS CREATED ON 04-21-1987)
(THE ROUTINE READIT HAS NOT BEEN CHANGED) (
THE ROUTINE PRINTIT WAS CREATED ON 04-21-1987) (THE ROUTINE
PRINTIT HAS NOT BEEN CHANGED))
```

Figure A6: History Manuals Tedit Window



MOVE CURSOR TO THIS WINDOW

PRESS THE MIDDLE MOUSE BUTTON TO DISPLAY MENU WITH SELECTIONS

Figure A7: Edit Window & Menu

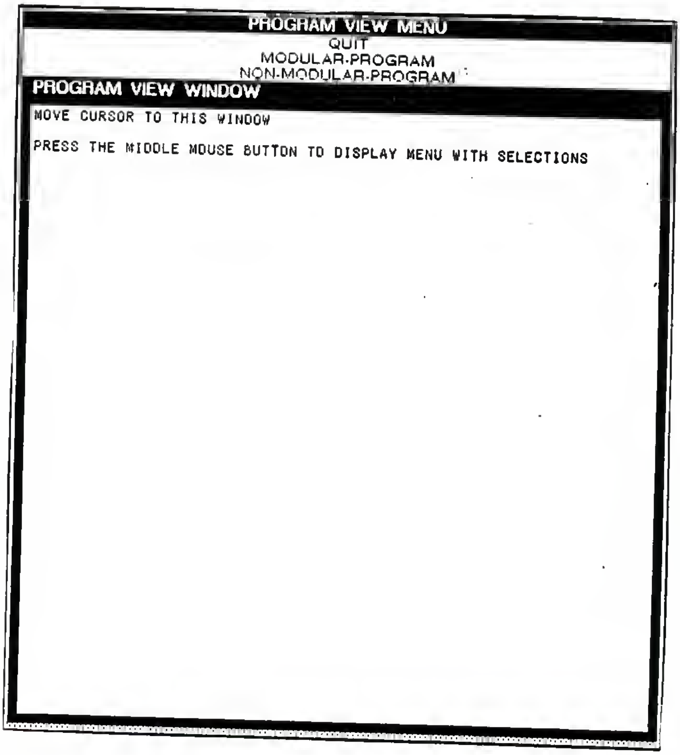
Variable and Module Name Search Feature

The final menu item available in the user prompt menu is Search Program. See figure A1. This item allows the user to search the program for module or variable names and other information based on if the program is modular in construct or if the program is nonmodular in construct. If the program is modular then both module and variable names can be searched for, and if the program is not modular in construct only variable names can be searched for.

When a user selects the Search Program item of the user prompt menu a new program view window is displayed. The user must move the mouse cursor to the window and click the middle mouse button to display the program view menu. See figure A8. The program view menu has the following items: Quit which closes the program view window and redisplays the user prompt window, Modular program which allows the user to search for both module and variable names, and finally the Nonmodular Program which allows the user to search for variables in the program being maintained.

When the Modular Program item is selected from the program view menu a search window is displayed and the user can move the mouse cursor to the new window and can click the middle mouse button to display the search menu. See figure A9. The search menu has the following items: Quit which closes the search window and redisplays the program view window, Search for Modules which will search for modules called by the program being maintained, and finally Search for Variables which determines what variable names are used by the program being maintained.

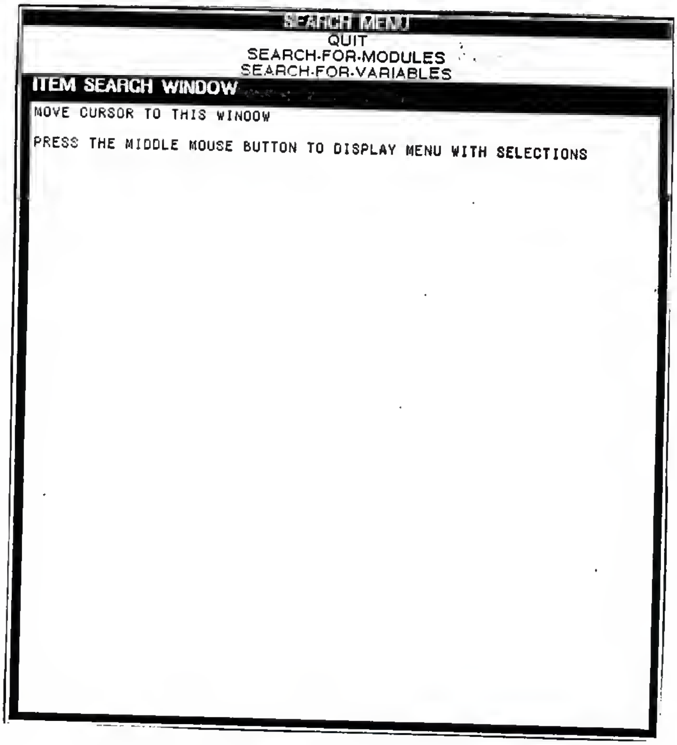
When the search for modules item is selected MAT1 will display a graph containing all modules or subroutines that the program calls as leaf nodes and the program name as the root node in the graph. See figure A10. The user will need to select a



MOVE CURSOR TO THIS WINDOW

PRESS THE MIDDLE MOUSE BUTTON TO DISPLAY MENU WITH SELECTIONS

Figure A8: Program View Window & Menu



```
MOVE CURSOR TO THIS WINDOW

PRESS THE MIDDLE MOUSE BUTTON TO DISPLAY MENU WITH SELECTIONS
```

Figure A9: Search Window & Menu

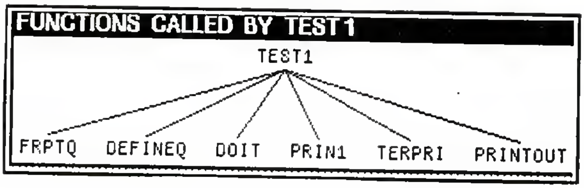


Figure A10: Functions Called Graph

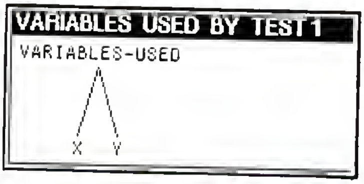


Figure A11: Variables Used Graph

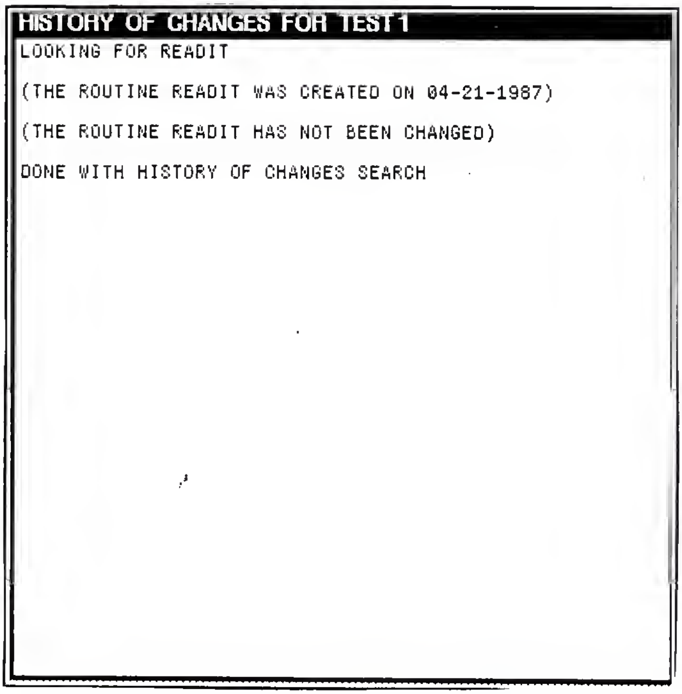
position on the screen for the graph. This is done by moving the graph shadow to the desired screen location and pressing the middle mouse button to display the graph.

This graph can now be used to obtain more information about the program. By moving the mouse cursor to a desired node in the graph and pressing the left mouse button a new graph will be displayed showing all subroutines or modules called by the selected module. See figure A10. This new graph has the selected node as the root node and all subroutines or submodules as leaf nodes in the graph tree. If no subroutines or modules are called by a module then an appropriate message is displayed in the search window.

The use of the middle mouse button to select a node from the original graph will display all variables used by a particular node in the graph in a new variables used graph. See figure A11. If the module uses no variables an appropriate message is displayed in the search window.

The variables used graph can now be used to obtain further information about the program being maintained. By selecting a leaf node in the variables used graph with the left mouse button, all history of changes pertaining to that variable is displayed in a window that is created by this action. See figure A12. The use of the middle mouse button in the variables used graph will display all the documentation that is available for the variable in a documentation display window. See figure A13.

The selection of Search for Variables from the search menu will produce the variables used graph with the same options that are discussed above. See figure A11. Selections of incorrect nodes in all graphs will display error messages in the correct display windows.



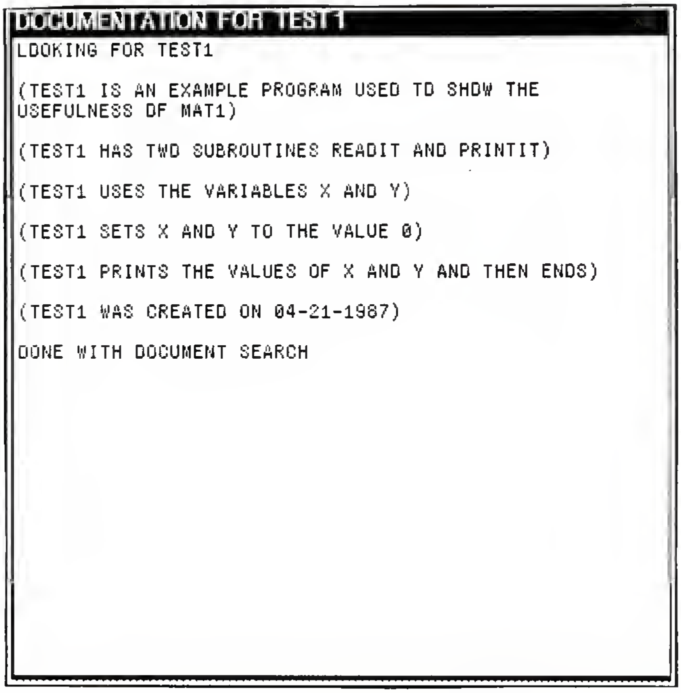
```
HISTORY OF CHANGES FOR TEST1
LOOKING FOR READIT

(THE ROUTINE READIT WAS CREATED ON 04-21-1987)

(THE ROUTINE READIT HAS NOT BEEN CHANGED)

DONE WITH HISTORY OF CHANGES SEARCH
```

Figure A12: History of Changes Display



```
DOCUMENTATION FOR TEST1
LOOKING FOR TEST1

(TEST1 IS AN EXAMPLE PROGRAM USED TO SHOW THE
USEFULNESS OF MAT1)

(TEST1 HAS TWO SUBROUTINES READIT AND PRINTIT)

(TEST1 USES THE VARIABLES X AND Y)

(TEST1 SETS X AND Y TO THE VALUE 0)

(TEST1 PRINTS THE VALUES OF X AND Y AND THEN ENDS)

(TEST1 WAS CREATED ON 04-21-1987)

DONE WITH DOCUMENT SEARCH
```

Figure A13: Documentation Display

The selection of the Nonmodular Program item from the program view menu will open a nonmodular search window. The nonmodular search window has a menu associated with it that has two items to select from. These are: Quit which closes the nonmodular search window and redisplays the program view window, and Search for Variables which produces the same results as does the search for variables item in the modular search menu. See figure A11.

In all cases with any window or graph that is displayed the use of the right mouse button will allow the maintainer the standard Interlisp-D right mouse button functions that deal with windows. These options are discussed in the manuals for the 1186 and in the Interlisp-D primer. MAT1 does not close all windows so the use of the right mouse button becomes important in avoiding the clutter caused by having too many windows being displayed on the screen at one time. Do not close the user prompt window or any window created by the user prompt menu with the right mouse button. A user can close graph windows and documentation and history display windows with the right mouse button, but do so only if all information in the window has been read and understood.

INTERLISP-D CAUTIONS

Some cautions must be noted. When a user changes a floppy disk it is necessary to enter the command (FLOPPY.WAIT.FOR.FLOPPY) in the tty window when the caret is blinking. This command informs the Interlisp-D environment that a new disk is being used. If this command is not entered the contents of the new (second) floppy disk may be destroyed. At times, when you exit from MAT1 the caret may not be blinking in the tty window. When this occurs, it is necessary to do a Control-D and then enter the command (TTYDISPLAYSTREAM TTY). This command will redirect the display to the tty window. The Control-D will cause a break window to appear in Interlisp-D. The control key is marked EDIT and is on the lower left hand portion of the 1186. When a user needs to restart the Maintenance Assistance Tool, two commands need to be entered to do this. The first command creates an instance of the maintenance object and the second sends the name of the method that is to be started to the instance that was created. These two commands are: (<- $Maintenance New 'M1) and (<- $M1 StartUp). The case of the letters is critical in the issuance of the send commands in LOOPS so these commands must be entered in the form that the example shows. For a further explanation of MAT1 see the Technical manual.

Appendix B

TECHNICAL MANUAL

The LOOPS programming environment provides an ideal working set of functions for developing knowledge based tools that use object oriented programming techniques. Since LOOPS is based on the Interlisp-D programming language, all of the special features of Interlisp-D can be incorporated into the Maintenance Assistance Tool (MAT1). Some of the features that Interlisp-D supports include: windows and menus, graphs, and the use of the Masterscope facility. Interlisp-D also provides logo windows, easy entry of information from the keyboard, record constructs, and many built in list handling and file handling functions that are necessary for the actions that MAT1 performs.

INTERLISP-D FEATURES

MAT1 relies on the use of the 1186 mouse. By incorporating windows, menus, and the mouse, the use of MAT1 is simplified almost to the point of move and click to perform the functions provided by the tool. The Maintenance Assistance Tool uses windows to prompt the user for information and to display messages and text. Menus are used to display functions available while using the tool. Some information about the program being maintained is displayed graphically. These graphs can then be used to display more information about a program that is being maintained.

PROGRAM STRUCTURE

The Maintenance Assistance Tool uses the LOOPS class inheritance lattice shown in figure B1. The root object or class is called Maintenance. This class has a method

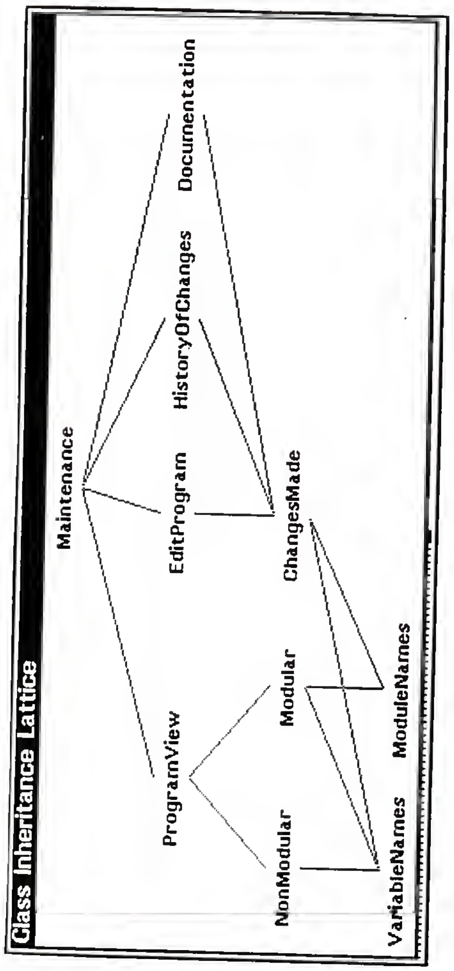


Figure B1: LOOPS Class Inheritance Lattice

called StartUp which is used to display the opening or user prompt window. This window is defined by setting the variable Uwindow to the correct createw parameters. A menu is attached to the top of the user prompt window when a user moves the mouse cursor to the user prompt window and then clicks the middle mouse button. The programming technique that is used to attach a menu to a window is performed by defining a window property called button event function (buttoneventfn) using the window property command. Normally, the button event function is Totop which brings a window to the uppermost display level of the screen (in other words no other windows cover it). By defining a function called Menuup to be the button event function and limiting the button on the mouse to be Only Middle, when a user clicks the middle mouse button while the cursor is inside the correct window, the menu will be displayed attached to the top of the correct window. This same method of attaching windows and menus is used in each method that displays a menu on top of a window in MAT1.

In order to define a menu in Interlisp-D, a variable is set to the correct create menu parameters. The title of the menu determines what text is displayed in the title block of the menu window. The items portion of the menu record determine which items will be displayed in the menu so that the user can select them. The centerflg field centers all menu items in the window created for the menu. The whenheld field determines what function is called when a user moves the mouse cursor to a menu item and then presses and holds the left or middle mouse buttons.

The whenheld function that is invoked will then display information about the item that was selected by holding the mouse button down. This information is displayed in the prompt window. The prompt window is the black window at the top of the screen in the left hand corner. The command Promptprint is used to print a message about the use of each menu item in the prompt window.

The whenselected field of a menu is used to determine which functions will be invoked when a user clicks the middle or left mouse button while pointing to a menu item with the mouse cursor. In this case, when a user does not want to select an item or to display information about the menu items, move the mouse cursor out of the menu window and release the pressed mouse button. This action will invoke the Progn function of the whenheld or whenselected functions. The Progn functions are defined as nil so that nothing will be done when no menu item is selected. The Progn function is critical to the correct use of menus, because if it is omitted the last function in the whenheld or whenselected functions are always invoked even when the mouse button is moved out of the menu window before a mouse button is released. The menu definition described above is used throughout MAT1 for displaying menus, their titles, and for determining what action is taken when a user uses the left or middle mouse buttons in the menu window.

Classes and Methods

The method StartUp not only displays an opening or user prompt window with a menu, it also displays a logo window that is used to show the name of the tool and the author. The logo window is a special form of the window record in Interlisp-D. The logo window displays file folder icons along with the name in the window.

StartUp also prompts the user to enter their first name and to enter the name of the program that is to be maintained. By redirecting the tty display stream to the correct window, information can be entered in windows other than the tty window.

The command Ttydisplaystream is used to redirect program input from the keyboard to the correct windows.

All messages displayed by MAT1 are printed using the Printout function. In using this function the window that the information is to be displayed in is specified by using the variable name that holds the definition of the window. The format item .skip is used to provide horizontal line spacing for the messages. Finally, the message to be displayed is inclosed in double quotes.

The method StartUp invokes new methods depending on which items are selected from the menu provided at the top of the user prompt window. In order to invoke a new method a message is passed to the class that contains the method creating a new instance of the class. A message is then passed to the instance of the class telling it which method to start. Four subclasses in the LOOPS hierarchy were developed to accommodate the necessary methods. The classes include HistoryOfChanges, Documentation, ProgramView, and EditProgram.

The function Quit.Stmenu is used to exit from the tool back to the Interlisp-D environment. The function See.Doc is used to invoke the method DisplayDocumentation in the class Documentation. The function See.Hst is used to invoke the method CheckHistory in the class HistoryOfChanes. The function Edit.Pgm is used to invoke the method EditTheSelectedProgram in the class EditProgram. The function Srch.Pgm is used to invoke the method ViewOfProgram in the class ProgramView. The Functions Stmenu.Whenheld and Stmenu.Whenselected are used for the whenheld and when-selected functions respectively.

The class HistoryOfChanges has three methods associated with it. These are: CheckHistory, EnterHistory, and LookForChanges. The method CheckHistory is used to display the history of changes window. This window definition is associated with the variable Hwindow and the window property button event function is set to display the history menu when the middle mouse button is pressed as was discussed in the opening menu explanation.

The method CheckHistory prompts the user to enter the disk drive that the history of changes files are resident on. A D is used to represent the hard disk drive and an F is used to represent the floppy disk drive. If there is a problem with the floppy drive the method exits and returns back to the user prompt window. The function Quit.Hmenu will exit from the history window to the user prompt window. The function Enter.Hst calls the method EnterHistory which allows the user to enter history of changes text into a file. The function Hst.Date prompts the user to enter a date on which changes were made to the program being maintained. It then calls the method LookForChanges and passes it the date to find and the name of the file to search. The function Hst.Purop prompts the user to enter a word that defines what purpose a module or variable has. It then calls the method LookForChanges passing the word to look for and the name of the file to search. The function Hst.All displays a history file in a Tedit window. The user can read all history information and can edit the information if it is so desired. The functions Hmenu.Whenheld and Hmenu.Whenselected are performed when an item in the history menu is pointed to and the left or middle mouse button is pressed and held or if the buttons are clicked on a menu item. The function Hmenuup displays the history menu attached to the history window on the top.

The method EnterHistory prompts the user to enter an answer as to if there are existing history files for the program being maintained. If history files exist the function Dooldhfile is invoked and if no history files exist the function Donewhfile is called.

The function Dooldhfile reads in the old history of changes information and assigns that information to the variable Filestuffin. This information is displayed in the history of changes window. Next, the function Inputhst is called passing it the old file information. Inputhst calls the recursive function Omt. Omt calls the function Readhst which reads information from the tty window that a user enters. The information is CONSed to a list that becomes a sentence. In order to stop the input if information the user must press the space bar and enter a period, question mark, or an exclamation point. When this occurs the user is then asked if they want to enter more history information. If a user responds with a Y the function Readhst is called again otherwise the history information is returned to the Dooldhfile function where it is stored on disk and redisplayed in the history of changes window.

The function Donewhfile operates in the same manner that the old history file function does except that no old information is passed to the Inputhst function. In either case the list of history information must be reversed since the CONS operation adds information to the front of a list. In order for the information to appear to be in the correct order it must be reversed or switched end for end using the reverse command.

The method LookForChanges is used to search the history files for a specified pattern. The history files are stored in a large list containing sentences or blocks of information which themselves are lists. The function Hlookatlist is called passing to it the list to search and the pattern to search for. The CAR of the list (the first sentence) is then obtained and this information along with the pattern to find is passed to the function Hismemberof. Hismemberof checks to see if the pattern to be found is a member of the list passed to it. This is accomplished by using the Eqmemb command. If the pattern is a member of the list the list is then displayed in the history display window that the method creates. Hlookatlist is a recursive function that keeps calling itself with the remainder of the history list (the CDR) until nothing is left in the list to search.

The class Documentation has three methods associated with it just as does the HistoryOfChanges class. Three methods are: DisplayDocumentation, EnterDocumentation, and LookForDocuments. Each of these three methods correspond to the methods described for the history of changes class. The method DisplayDocumentation corresponds to CheckHistory except that the documentation can be searched for by a note number and the history cannot. The method EnterDocumentation corresponds to the method EnterHistory. Finally, the method LookForDocuments corresponds to the method LookForChanges. Each corresponding method operates in the same manner that was described for the HistoryOfChanges methods.

The class EditProgram has one method called EditTheSelectedProgram. This method displays the edit window and attaches the edit menu to the edit window. The function Quit.Emenu closes the edit window and redisplays the user prompt window. The function Edit.Prog invokes the method MakeChanges in the class ChangesMade. The functions Emenu.Whenheld and Emenu.Whenselected are called when the left or middle mouse button is either pressed and held or is clicked while the mouse cursor is pointing to a menu item.

The class ChangesMade has one method called MakeChanges. This method is used to prompt the user to enter an F if the program to be edited is on the floppy disk or to enter a D of the program being maintained is on the hard disk drive. The user should select FNS from the pop up menu that Dedit causes to appear. A Dedit window will then open with the function definitions for the program being maintained.

The class ProgramView has one method called ViewOfProgram. This method is used to display the program view window and the program view menu. The function Quit.Pvmenu is used to close the program view window and to display the user prompt window. The function Pv.Modular is used to invoke the method ViewMod in the class Modular. The function Pv.Nonmodular is used to invoke the method ViewNonMod in the class NonModular. The functions Pvmenu.Whenheld and Pvmenu.Whenselected are used to determine program action when the left or middle mouse buttons are held or clicked, while the mouse cursor is pointing to a menu item.

The class NonModular has one method called ViewNonMod. This method is used to display the nonmodular search window and its attached menu. The function Quit.Nonmodsearchmenu is used to close the nonmodular search window and to redisplay the program view window. The function Nmsearch.vars is used to invoke the method SearchForVariables in the class VariableNames. The Whenheld and When-selected functions act in the manner described previously.

The class Modular has one method called ViewMod. This method is used to display the modular search window and its attached menu. The function Quit.Searchmenu is used to close the modular search window and to redisplay the program view window. The function Search.Mods invokes the method SearchForModules in the class ModuleNames. The function Search.Vars invokes the method SearchFor-Variables in the class VariableNames. Again, the whenheld and whenselected functions act in the manner previously described.

The class ModuleNames has one method called SearchForModules. This method is designed to use the Masterscope facility of Interlisp-D. The information returned by masterscope is then displayed in graph form. The graph then becomes a 'menu' for obtaining more information about the program being maintained. The Masterscope command Who Does XXX Call is used, where XXX is the name of the program being maintained. The answer returned by masterscope is a list containing all the functions or modules that the program calls. If no functions are called by the program then an appropriate message is displayed in the modular search window, otherwise a graph is created using the function Layoutsexpr with the program name as the root node and the answer to the masterscope question as leaf nodes. The graph also has left and middle mouse button functions associated with it. The graph must be moved to a desired location on screen and the middle mouse button should be clicked to display it.

Once the graph is displayed it can now be used as a menu to display more information about the program currently being maintained. By moving the mouse cursor to a leaf node in the graph and pressing the left mouse button the function called Lmbfn is invoked. This function displays another graph using the node that was selected as the root node and the answer to the masterscope query Who Does XXX Call, where XXX is the node that was selected from the original graph. This allows the user to display all subroutines or functions that a module of the original program calls.

By using the middle mouse button to select a node in the original graph, the function Mmbfn is called. This function is used to display a graph that contains all the variables used by a particular module or the program. The root node is called Variables-used and the leaf nodes are the members of the list returned by the masterscope question Who Does XXX Use, where XXX is the node that was selected from the original graph.

The variables used graph can be used to obtain more information about the variables. By selecting a leaf node in the variables used graph with the left mouse button, the function Modnlmbfn is invoked. This function calls the history of changes display method LookForChanges in the class HistoryOfChanges. Selection of a leaf node in the variables used graph with the middle mouse button calls the function Modnmmbfn. This function calls the documentation display method LookForDocuments in the class Documentation passing it the variable name to search for.

If Masterscope returns an answer with no values i.e. nil, the program does not display a new graph. A message stating that no variables are used or that no functions are called is printed in the appropriate display window. If a user makes a bad selection in the graph, the message choose another node is displayed, prompting the user to make another node selection.

The class VariableNames has one method called SearchForVariables. This method is used to display a graph containing the answer to the masterscope question Who Does XXX Use, where XXX is the program name. The user can then use the left mouse button to see what modules use a variable selected from the original variables used graph. The answer to the masterscope question Who Uses XXX , where XXX is the variable is displayed in a new graph. The root node of the graph if Functions-That-Use and the leaf nodes are the answer to the masterscope query described above. The left mouse button can be used to select nodes in the function that uses graph. This will cause the method LookForChanges in the class HistoryOfChanges to be invoked. All Changes for the selected node will be displayed in the history window that is created. The middle mouse button can be used to invoke the method LookForDocuments in the class Documentation. All Documentation containing the word that is the selected node will be displayed in a documentation window. SearchForVariables has no middle mouse button function defined for use in the original variables used graph.

The function Packfilename is used extensively throughout MAT1 to construct the correct file name for editing and for searching history and documentation files. In all of the mouse button functions defined for a graph, both the node that was selected and the window that contains the graph are passed to the mouse button function. This is a convention of Interlisp-D. In all cases of creating a graph, a window, or a menu, the definitions must be stored in a variable using the Setq function. This action creates an instance of these definitions that can be used over and over in the program. The use of the if statement in Interlisp-D is like the use of the Cond statement, so it is necessary to have an escape clause if no correct answer is found. All the correct forms of programming statements can be found in the manuals accompanying the Xerox 1186 workstation.

Appendix C

MAT1 Source Code

```
(OSK)<LISPFILES>AL>TOOLINIT.;2   B-May-87 11:12:24  Page: 1

(PRETTYCOMPRINT TOOLINITCOMS)

(RPAQQ TOOLINITCOMS ((FNS TOOLINIT)))
(OEFINEQ

(TOOLINIT
  [LAMBOA NIL                                                    (* edited: " 8-May-87 11:11")
    (SETQ FIRSTNAME (QUOTE AL))
    (GREET)
    (CNDIR (QUOTE (DSK)<LISPFILES>AL>))
    (LOAO (QUOTE TEST1))
    (TEST1)
    (LOAO (QUOTE (OSK)<LISPFILES>LIBRARY>GRAPHER.OCOM))
    (FILESLOAO (QUOTE MAINTENANCE)
               (QUOTE STARTUP)
               (QUOTE OOCUMENTATION)
               (QUOTE HISTORYOFCHANGES)
               (QUOTE EOITPROGRAM)
               (QUOTE PROGRAMVIEW)
               (QUOTE OISPLAYOOCUMENTATION)
               (QUOTE CHECKHISTORY)
               (QUOTE EOITTHESELECTEOPROGRAM)
               (QUOTE VIEWOFPROGRAM)
               (QUOTE CHANGESMAOE)
               (QUOTE MOOULAR)
               (QUOTE NONMOOULAR)
               (QUOTE MOOULENAMES)
               (QUOTE VARIABLENAMES)
               (QUOTE BM1)
               (QUOTE BM2))
    (_ $Maintenance New (QUOTE M1))
    (_ $M1 StartUp])
)
(OECLARE: OONTCOPY
  (FILEMAP (NIL (264 1149 (TOOLINIT 274 . 1147)))))
STOP
{OSK}<LISPFILES>AL>MAINTENANCE.;1  31-Mar-87 16:02:48  Page: 1


(PRETTYCOMPRINT MAINTENANCECOMS)

(RPAOO MAINTENANCECOMS ((CLASSES Maintenance)))
(DEFCLASSES Maintenance)
|OEFCLASS Maintenance
   (MetaClass Class Edited:                                    (* edited: "31-Mar-87 14:19"))
   (Supers Object)
   (InstanceVariables (Pgm Nil doc                             (* IV added by ))
                      (Name Nil doc                            (* IV added by ))))]

(OECLARE: OONTCOPY
  (FILEMAP (NIL)))
STOP
```

```
(DSK)<LISPFILES>AL>STARTUP.;1  22-Apr-87 D9:56:D4  Paga: 1


(PRETTYCDMPRINT STARTUPCDMS)

(RPAQO STARTUPCOMS ((METHODS Meintenanca.StertUp)))
[METH Meintananca  StertUp NIL
      (* Naw method tamplata)]


(DEFINEQ

(Meintenence.StertUp
  (Method ((Maintenence StartUp)
          salf)
                                                          (* aln "22-Apr-87 D8:32")
                                                          (* Naw method templeta)

(* Tha method StartUp is used to displey an opaning window. This window hes e menu attached to it that allows the
usar to anter and search for documantetion, antar end saarch for e history of changes, adit the selected progrem,
end search tha progrem for modula and variabla nemas. Tha method prompte tha usar for their name end for the
program to ba workad on. Tha usar must move the cursor to the opaning window and click the middle mouse button to
bring up the manu. Once tha manu has been displeyad, to invoka tha dasired function move the cursor to the dasired
function and click tha left or middla mouse button.)


(* Tha function OUIT.STMENU is used to exit from MAT1. This function reseta tha opening window's buttonevant
function end closas the opaning window. Tha output stream is than returned to the tty window.)


(DEFINEO (OUIT.STMENU NIL (WINDDWPROP UWINDDW (OUOTE BUTTONEVENTFN)
                                               (FUNCTION MENUUP))
                          (CLOSEW UWINDOW)
                          (TTYDISPLAYSTREAM TTY)))
                                                  (* Tha function SEE.DDC starts the documentation enter
                                                   end display function of MAT1.)
(DEFINEO (SEE.DDC NIL (_ ($ Documantation)
                             New
                             (OUOTE DC1))
                      (_ ($ DC1)
                         DispleyDocumentation)))
                                                  (* The function SEE.HST starts tha history of changas
                                                   enter end displey function of MAT1.)
(DEFINEO (SEE.HST NIL (_ ($ HistoryOfChanges)
                             New
                             (QUOTE HST1))
                      (_ ($ HST1)
                         CheckHistory)))          (* The function EDIT.PGM starts the editor function of
                                                   MAT1.)
```

```
(DEFINEQ (EDIT.PGM NIL (_ ($ EditProgram)
                          New
                          (QUOTE EP1))
                       (_ ($ EP1)
                          EditTheSelectedProgram)))
                                              (* The function SRCH.PGM starts the searching function
                                                 of MAT1.)
(DEFINEQ (SRCH.PGM NIL (_ ($ ProgramView)
                          New
                          (QUOTE PV1))
                       (_ ($ PV1)
                          ViewOfProgram)))        (* The function STMENU.WHENHELD displays a description
                                                 of what each selection in the startup menu does.)
(DEFINEQ (STMENU.WHENHELD (ITEM.SELECTED MENU.FROM BUTTON.PRESSED)
                          (SELECTQ ITEM.SELECTED
                                   (QUIT (PROMPTPRINT

                                      "EXIT FROM THE MAINTENANCE ASSISTANCE TOOL"))
                                   (SEE-DOCUMENTATION (PROMPTPRINT
                                 "DISPLAYS DOCUMENTATION FOR THE SELECTED PROGRAM"))
                                   (SEE-HISTORY (PROMPTPRINT
                         "DISPLAYS THE HISTORY OF CHANGES FOR THE SELECTED PROGRAM"))
                                   (EDIT-PROGRAM (PROMPTPRINT
                                                      "EDIT THE SELECTED PROGRAM"))
                                   (SEARCH-PROGRAM (PROMPTPRINT
                  "SEARCHES FOR MODULE OR VARIABLE NAMES IN THE SELECTED PROGRAM"))
                                   (PROGN NIL))))
                                             (* The function STMENU.WHENSELECTED defines which
                                                function described above, to invoke when the user
                                                selects one of the options from the startup menu.)
(DEFINEQ (STMENU.WHENSELECTED (ITEM.SELECTED MENU.FROM BUTTON.PRESSED)
                              (SELECTQ ITEM.SELECTED
                                       (QUIT (QUIT.STMENU))
                                       (SEE-DOCUMENTATION (SEE.DOC))
                                       (SEE-HISTORY (SEE.HST))
                                       (EDIT-PROGRAM (EDIT.PGM))
                                       (SEARCH-PROGRAM (SRCH.PGM))
                                       (PROGN NIL))))

(* The function MENUUP is used to display the startup menu when the middle mouse button is pressed and the cursor
is in the startup window. It uses attachwindow to attach the menu to the top of the startup window.)
{DSK}<LISPFILES>AL>STARTUP.;1  22-Apr-87 09:56:04  Page: 2
(DEFINEQ (MENUUP (WINDOWNAME)
                 (if (MOUSESTATE (ONLY MIDDLE))
                     then (ATTACHWINDOW STMENU UWINDOW (QUOTE TOP)))
                 (CLOSEW LOGO)))
```

```
(* The verieble STMENU holds the menu creeted from the proper record definition. The field title is the title for
the menu. The field items defines the allowed options for the menu. Centerflg centers the options in the menu.
Whenheld and Whenselected tell the menu whet to do when the middle or left mouse buttons are either held down or
are let up i.e. selected)


(SETQ STMENU (MENUWINOOW (creeta MENU
                                       TITLE _ "STARTUP MENU"
                                       ITEMS _ (QUOTE (QUIT SEE-OOCUMENTATION SEE-HISTORY
                                                      EOIT-PROGRAM SEARCH-PROGRAM))
                                       CENTERFLG _ T
                                       WHENHELOFN _ (FUNCTION STMENU.WHENHELO)
                                       WHENSELECTEOFN _ (FUNCTION STMENU.WHENSELECTEO))
                               T))

(* The variable LOGO is used to store the information for the logo window that displeys Maintenence Assistence Tool
et the screen coordinates ecross 125 and up 5 from the lower left hand corner of the screen.)


(SETO LOGO (LOGOW "MAINTENANCE ASSISTANCE TOOL"
                  (create POSITION
                          XCOORO _ 125
                          YCOORO _ 5)
                 "AL NICHOL'S MAINTENANCE ASSISTANCE TOOL         APRIL 7, 1987"))
                                             (* The verieble UWINOOW holds the definition of the
                                              startup window. The title of the window is user prompt
                                              window.)
(SETO UWINOOW (CREATEW (QUOTE (500 200 500 500))
                      "USER PROMPT WINOOW" 10))

(* Setting WINOOWPROP tells lisp that when the cursor is moved to the window UWINOOW it will cell the function
menuup when the middle mouse button is clicked.)
{OSK}<LISPFILES>AL>STARTUP.;1  22-Apr-87 09:56:04  Page: 3
(WINOOWPROP UWINOOW (QUOTE BUTTONEVENTFN)
               (FUNCTION MENUUP))

(* The remeining code prompts the user for their name and the neme of the program to work on.
It elso instructs the user to move the mouse cursor to the new window and click the middle mouse button to displey
the startup menu.)


(PRINTOUT UWINOOW "PLEASE TURN THE CAPS LOCK ON")
(PRINTOUT UWINOOW .SKIP 2 "ENTER YOUR FIRST NAME! ")
(TTYOISPLAYSTREAM UWINDOW)
(_@
  :Neme
  (REAO))
```

```
(TTYOISPLAYSTREAM TTY)
(PRINTOUT UWINOOW .SKIP 1 "HELLO " (@ :Name)
           " GLAO TO HAVE YOU ABOARO")
(PRINTOUT UWINOOW .SKIP 2 "WHICH PROGRAM WOULO YOU LIKE TO WORK ON TOOAY? ")
(TTYOISPLAYSTREAM UWINOOW)
(_@
  :Pgm
  (REAO))
(TTYOISPLAYSTREAM TTY)
(PRINTOUT UWINOOW .SKIP 1 "WE ARE WORKING ON PROGRAM " (@ :Pgm))
(PRINTOUT UWINOOW .SKIP 2 "MOVE THE CURSOR TO THIS WINOOW")
(PRINTOUT UWINOOW .SKIP 2 "PRESS THE MIOOLE MOUSE BUTTON TO OISPLAY MENU WITH SELECTIONS")
(PRINTOUT T)))
)
(OECLARE: OONTCOPY
  (FILEMAP (NIL (3B9 7431 (Maintenance.StartUp 399 . 7429)))))
STOP
```

```
(DSK)<LISPFILES>AL>DOCUMENTATION.;1   8-May-87 1D:54:38  Paga: 1


(PRETTYCOMPRINT DOCUMENTATIONCOMS)

(RPAOO DOCUMENTATIONCOMS ((CLASSES Documentation)
                          (METHODS Documentation.EntarDocumentation Documentation.LookForDocumants))
)
(DEFCLASSES Documentetion)
[DEFCLASS Documentation
   (MetaClass Class Edited:                                    (* edited: "31-Mar-87 14:20"))
   (Supars Maintenance)
   (InstanceVariablas (DirName NIL doc                         (* IV added by))
                      (FileNema NIL doc                        (* IV added by))
                      (WhatToFind NIL doc                      (* IV added by)))]

[METH Documentation  EntarDocumentation NIL
      (* New method template)]


[METH Documentation  LookForDocumenta (WhatToLookFor FileToSaarch)
      (* New method tamplate)]


(DEFINEO

(Documentation.EnterDocumentation
  (Method ((Documentation EntarDocumentetion)
           self)                                               (* aln " 8-Mey-87 1D:24")
                                                               (* New method tamplata)

(* Tha method EnterDocumentation allows the user to anter documantation for tha program being maintained.
The documentetion Is entered as a list of lists, with each sublist containing a singla sentenca or block of
information. In order to stop entering documentetion the user must presa tha space bar and a pariod, question mark,
or exclamation point. This causes tha program to store the sentenca or block as a list by CDNSing aach word onto a
singla list. The user is then prompted to anter more documentation. Thay can answar Y or N to aither enter mora
documentation or to stop tha antry procass. The method querias tha usar as to the existance of documentation files
and acts accordingly.)


(* The function INPUTDDC prompts tha user to entar documantation in tha tty window and than calls tha recursive
function onemoretime in order to store the informetion that was typed in as a list. In order to stop documentation
entry the user must press the space bar and then either a period. question merk. or exclamation point.)


(DEFINEO (INPUTDDC (FLIN)
                   (SETQ CDMBLOCK FLIN)
                   (CLEARW DWINDOW)
```

```
                    (PRINTOUT OWINOOW "ENTER THE NEW OOCUMENTATION IN THE TTY WINOOW")
                    (PRINTOUT OWINOOW .SKIP 2
                         "WHEN YOU ARE OONE TYPING A SENTENCE OR BLOCK OF COMMENTS")
                    (PRINTOUT OWINOOW .SKIP 2
                           "PRESS THE SPACE BAR ANO FOLLOW WITH A . ? OR !")
                    (PRINTOUT OWINOOW .SKIP 2
                        "ONE OF THESE THREE CHARACTERS WILL STOP THE ENTRY FUNCTION")
                    (PRINTOUT T)
                    (ONEMORETIME COMBLOCK)))

(* The function REAOOOC is a recursiva function that reads tha information that is typed in tha tty window and
recursively CONSs tha information onto a sentence list. If a period, exclamation point, or question mark is found
the consing to the current list stops.)


(OEFINEQ [REAODOC (X)
                   (SETQ X (REAO))
                   (if (OR (EQ X (QUOTE %.))
                           (EQ X (QUOTE !))
                           (EQ X (QUOTE ?)))
                      then NIL
                    else (SETQ X (CONS X (REAOOOC X])

(* The function ONEMORETIME is a recursive function that allows the user to enter more than one santance or
documentation block et a time. The function CONSs tha santence list that was constructed by READOOC into a list of
lists that can ba used by the search facility for the documantation saarch. Commout holds the constructad sentence
list and Comblock is the list of sentence lists. If tha user wants to anter another sentence they must raspond with
a Y and then continue typing documentation.)
{OSK}<LISPFILES>AL>OOCUMENTATION.;1   B-May-B7 10:54:38   Page: 2
(OEFINEO (ONEMORETIME (COMBLOCK)
                     (SETQ COMMOUT (REAOOOC))
                     (SETQ COMBLOCK (CONS COMMOUT COMBLOCK))
                     (PRINTOUT T "MORE? (Y/N) ")
                     (SETQ ANS (REAO))
                     (if (OR (EQ ANS (QUOTE Y))
                             (EQ ANS (QUOTE y)))
                        then (ONEMORETIME COMBLOCK)
                      else (SETQ BIGBLOCK COMBLOCK))))

(* The function OONEWFILE is used to create a new documentetion file when none previously existed.
The function uses the progrem name with an extansion of .OOC as the file to store information.
The file is opened for output and the information to ba printed is placed in the variable stufftoprint which calls
the function INPUTOOC. The list of information to print is in reverse order because CONS appends to the front of a
list. The list is reversed and storad in the veriable STUFFOUT. Printout is used to write the list to the file.
The function then closas all open files and returns the displaystream to the tty window.)
(OEFINEO (DONEWFILE NIL (SETQ DNME (@ $DC1 :OirNeme))
                        (SETQ FN (@ $DC1 :FileNeme))
                        (OUTFILE FN)
                        (SETQ STUFFTOPRINT (INPUTDOC))
                        (SETQ STUFFOUT (REVERSE STUFFTOPRINT))
                        (PRINTOUT FN STUFFOUT)
                        (CLOSEALL T)
                        (TTYDISPLAYSTREAM TTY)))

(* The function DOOLDFILE is used to append new documentation to en existing documentation file.
The old documentetion file is read in and the information is stored in the verieble FILESTUFFIN.
The CAR of the informetion reed in is used since printout pleces the list producad by typing documentetion into
another list. The informetion is then reversad because to CONS eppends to the front of e list and to eppend to the
end you must reverse first. Once this is done the old file is eppended with the new documentation end the file is
then stored back to the specifiad disk directory. Finally, ell files are closed end the displeystreem is returned
to the tty window.)


(OEFINEO (DOOLDFILE NIL (SETQ ONME (@ $DC1 :OirNeme))
                        (SETO FN (@ $OC1 :FileName))
                        (INFILE FN)
                        (SETQ FILESTUFFIN (CAR (READFILE FN)))
                        (SETQ FSI (REVERSE FILESTUFFIN))
                        (PRINTOUT DWINDOW .SKIP 2 "THIS IS THE OLD DOCUMENTATION")
                        (PRINTOUT OWINOOW .SKIP 2 FSI)
                        (CLOSEALL T)
                        (OELFILE FN)
                        (OUTFILE FN)
                        (SETQ STUFFTOPRINT (INPUTOOC FSI))
                        (SETQ STUFFOUT (REVERSE STUFFTOPRINT))
                        (PRINTOUT OWINOOW .SKIP 2 "THIS IS THE NEW DOCUMENTATION")
                        (PRINTOUT DWINDOW .SKIP 2 STUFFOUT)
                        (PRINTOUT FN STUFFOUT)

                        (CLOSEALL T)
                        (TTYDISPLAYSTREAM TTY)))   (* The remeining code prompts the user to see if a
                                                      documentation file exists end tekes appropriate
                                                      action.)
(CLEARW DWINDOW)
(PRINTOUT OWINDOW "IS THERE A DOCUMENTATION FILE FOR THIS PROGRAM? (Y/N) ")
(TTYDISPLAYSTREAM OWINDOW)
(SETO ANS (READ))
(TTYDISPLAYSTREAM TTY)
(if (OR (EO ANS (OUOTE Y))
           (EO ANS (OUOTE y)))
    then (DOOLDFILE)
  else (DONEWFILE))
(CLEARW DWINDOW)
(TTYDISPLAYSTREAM TTY)))
```

```
{DSK}<LISPFILES>AL>DOCUMENTATION.;1   8-May-87 10:54:38  Page: 3

(Documantation.LookForDocuments
  (Method ((Documentation LookForDocuments)
           self WhetToLookFor FileToSeerch)                    (* aln "22-Apr-87 D9:52")
                                                               (* New method template)

(* Tha method LookForDocumante is ueed to eeerch the documentation files for a specified pettarn.
The documentation files must be stored with sentencas in lists for the eearch to work. The function ie passed tha
information to search for vie the ergument WhetToLookFor end the file to saarch vie FileToSearch.)


(* The function ISMEMBEROF is used to seerch e list to sae if the pettern passed to it is a member of tha list.
If the pattern is found the list is printed. This function is pessed tha list in LISTTOSEARCH and tha patten to
find in SEARCHVALUE.)


(DEFINEQ (ISMEMBEROF (LISTTOSEARCH SEARCHVALUE)
                         (SETQ SV SEARCHVALUE)
                         (SETQ LTS LISTTOSEARCH)
                         (if (EQMEMB SV LTS)
                             then (PRINTOUT OOCWINDOW .SKIP 2 LTS)
                           elea NIL)))

(* The function LDOKATLIST is a recursive function that checks the list that is pessed to it.
If the CAR of the list is empty NIL then the liet is empty and the eearch is done. If the CAR of the list is not
empty tha CAR of the list ie paeeed to the ISMEMBEROF routine to search for the corract pettern.
The pattern to search for is hald in WTF which is set to tha value passad to the LODKATLIST ergumant WORDTOFIND.
The ergument LISTIN holde the list to seerch. By teking the COR of the saarch list the remaining sentences in the
documentation fila cen ba eaarched.)


(DEFINEQ (LDDKATLIST (LISTIN WORDTOFIND)
                         (SETQ WTF WORDTOFIND)
                         (SETQ ISIN (CAR LISTIN))
                         (if (EQ ISIN NIL)
                             than NIL
                           else (ISMEMBEROF ISIN WTF))
                         (SETQ LN (CDR LISTIN))
                         (if (EQ ISIN NIL)
                             then (PRINTOUT DDCWINDDW .SKIP 2 "DDNE WITH DOCUMENT SEARCH")
                           alee (LOOKATLIST LN WTF))))
                                                (* Tha verieble DOCWINDOW is used to hold the
(SETD DOCWINDOW (CREATEW (DUOTE (1D 1D 4DD 4DD))    definition of the documentetion display window.)
                           (CDNCAT "DDCUMENTATIDN FOR " (@ $MI :Pgm))
                           5))                  (* The remaining code reads the documentation file and
                                                 cells LODKATLIST passing thet function tha list to
                                                 search and what to look for.)
{DSK}<LISPFILES>AL>DOCUMENTATION.;1    8-May-87 10:54:38   Page: 4
            (CLEARW OOCWINDOW)
            (SETO FN FileToSearch)
            (INFILE FN)
            (SETO OOCLIST (CAR (READFILE FN)))
            (CLOSEALL T)
            (SETQ WTLF WhatToLookFor)
            (PRINTOUT DOCWINDOW "LOOKING FOR " WTLF)
            (LOOKATLIST DOCLIST WTLF)))
)
(OECLARE: OONTCOPY
  (FILEMAP (NIL (1006 10228 (Oocumentation.EnterDocumentation 1016 . 7307) (
Oocumentation.LookForDocuments 7309 . 10226)))))
STOP
```

```
{DSK}<LISPFILES>AL>OISPLAYOOCUMENTATION.;1   8-May-87 10:46:20  Page: 1


(PRETTYCDMPRINT DISPLAYDOCUMENTATIDNCDMS)

(RPAQQ DISPLAYDOCUMENTATIONCOMS ((METHOOS Oocumentation.OisplayDocumentation)))
[METH Oocumentation  DisplayOocumentation NIL
      (* New mathod tamplate)]


(DEFINEQ

(Oocumentation.OisplayOocumentation
  (Method ((Documentation OisplayDocumentation)
           self)                                                (* eln " 8-May-87 1D:22")
                                                                (* Naw method tamplate)

(* The function DOOERR is callad whan tha computar detacts an error associated with the floppy disk driva.
This function tells the usar to chack tha floppy driva and then calls OUIT.DMENU which exits to the startup
window.)


(OEFINEO (DODERR NIL (SETQ BADERR T)
                    (CLEARW OWINODW)
                    (PRINTOUT OWINOOW
                   "FLOPPY NOT IN ORIVE OR OOOR NOT CLOSEO.  PLEASE CHECK FLOPPY ORIVE!")
                    (PRINTOUT DWINDOW .SKIP 2
                              "THIS ERRDR CAUSES A RETURN TO THE STARTUP MENU")
                    (for ZZ from ] to 15DD do NIL)
                    (CLEARW DWINOOW)
                    (OUIT.OMENU)))

(* Tha function QUIT.DMENU is used to exit from the documentetion window to the startup window.
It resats the window proparty of tha documantation window and then closas it. The function than reopans tha startup
window so that the usar can continue to use MAT1.)


(OEFINEQ (QUIT.DMENU NIL (WINDOWPROP DWINDDW (QUOTE BUTTONEVENTFN)
                                             (FUNCTION OMENUUP))
                         (CLOSEW DWINDOW)
                         (OPENW UWINOOW)
                         (TTYDISPLAYSTREAM TTY)))
                                                    (* Tha function ENTER.DDC calls the documentetion
                                                     entry function.)
(DEFINEQ (ENTER.DDC NIL (_ ($ OC1)
                           EnterOocumentation)))

(* The function DDC.OATE calls the documantation saarch function end pesses the saerch the file to search end what
to look for. In this case the search string is a date entered in the form MM-DD-19YY. The function also prompts tha
usar that files are baing loaded by displaying a bitmep in the documentation window.)
```

```
(DEFINEO (DDC.DATE NIL (CLEARW DWINOOW)
                        (BITBLT BM1 NIL NIL DWINDOW 400 4DD 60 60)
                        (PRINTOUT OWINDOW "PLEASE WAIT!  LOADING DOCUMENTATIDN FILES")
                        (for ZZ from 1 to 1D00 do NIL)
                        (CLEARW DWINDOW)
                        (PRINTOUT OWINDDW "WHAT DATE WOULO YDU LIKE TO LDOK FOR?")
                        (PRINTDUT DWINOOW .SKIP 2
                                "ENTER DATE IN FDRM 'DO-MM-19YY' AND SPECIFY LEAOING ZERDS")
                        (PRINTOUT DWINDOW .SKIP 2 "ENTER THE DATE TO LOOK FOR! ")
                        (TTYDISPLAYSTREAM DWINDDW)
                        (_@
                          $DC1 :WhatToFind (REAO))
                        (TTYOISPLAYSTREAM TTY)

                        (_ ($ DC1)
                             LookForDocuments
                             (@ $DC1 :WhatToFind)
                             (@ $DC1 :FileName))
                        (TTYDISPLAYSTREAM TTY)))
```

(* Tha function DOC.PURDP calls the documentation search function and passes the search tha file to search and what to look for. In this case the search string is a word that will identify the purpose of a module. The function elso prompts the user that the files are being loaded by displaying a bitmap in the documentation window.)

{DSK}<LISPFILES>AL>OISPLAYDOCUMENTATION.;1   B-May-87 10:46:2D   Page: 2

```
(OEFINED (DDC.PURDP NIL (CLEARW DWINDDW)
                        (BITBLT BM1 NIL NIL DWINDDW 40D 4DD 60 60)
                        (PRINTDUT DWINOOW "PLEASE WAIT!  LOADING ODCUMENTATIDN FILES")
                        (for ZZ from 1 to 1000 do NIL)
                        (CLEARW DWINDDW)
                        (PRINTOUT DWINDDW "ENTER A WORD THAT OEFINES THE PURPDSE OF A MDDULE")
                        (PRINTOUT DWINDOW .SKIP 2 "ENTER THE PURPDSE WDRD! ")
                        (TTYOISPLAYSTREAM DWINDOW)
                        (_@
                          $OC1 :WhatToFind (READ))
                        (TTYDISPLAYSTREAM TTY)
                        (_ ($ OC1)
                             LookForDocuments
                             (@ $DC1 :WhatToFind)
                             (@ $DC1 :FileName))
                        (TTYDISPLAYSTREAM TTY)))
```

(* The funciton DOC.NOTES calls the documentation search function and passes the search the file to search and what to look for. In this case the search string is e note numbar that will identify salacted information about the program. The function also prompts tha user that files are being loaded by displaying a bitmap in the documentation window.)

```
(OEFINEO (OOC.NOTES NIL (CLEARW OWINOOW)
                       (BITBLT BM1 NIL NIL DWINOOW 400 400 60 60)
                       (PRINTOUT OWINDOW "PLEASE WAIT!  LOAOING DOCUMENTATION FILES")
                       (for ZZ from 1 to 1000 do NIL)
                       (CLEARW OWINOOW)
                       (PRINTOUT DWINDOW "ENTER A NOTE NUMBER TO LOOK FOR")
                       (PRINTOUT OWINOOW .SKIP 2 "THE FORM IS NOTE#. WHERE # IS AN INTEGER.")
                       (PRINTOUT OWINDOW .SKIP 2 "ENTER THE NOTE TO FINO! ")
                       (TTYOISPLAVSTREAM OWINOOW)
                       (_@
                         $OC1 :WhatToFind (REAO))
                       (TTYOISPLAVSTREAM TTY)
                       (_ ($ DC1)
                            LookForOocuments
                            (@ $OC1 :WhatToFind)
                            (@ $DC1 :FilaName))
                       (TTVOISPLAVSTREAM TTY)))

(* The function OOC.MANUALS displays all the current documentation for tha selacted program.
This function displays the documentation in a TEOIT window. Tha user can edit the documentation from this window if
desired. The function prompts the user that files are baing loaded and then displays a TEOIT window.)
{DSK}<LISPFILES>AL>DISPLAYDOCUMENTATION.;1   B-May-87 10:46:20   Page: 3

(OEFINEQ (OOC.MANUALS NIL (CLEARW DWINOOW)
                        (BITBLT BM1 NIL NIL OWINOOW 400 400 60 60)
                        (PRINTOUT OWINOOW "PLEASE WAIT!  LOADING OOCUMENTATION FILES")
                        [if (OR (EQ (@ $DC1 :OirName)
                                        (QUOTE 0))
                                  (EQ (@ $OC1 :OirName)
                                      (QUOTE d)))

                          then (TEOIT (PACKFILENAME (OUOTE BOOY)
                                                    (CONCAT "<LISPFILES>AL>"
                                                            (@ $M1 :Pgm)
                                                            ".OOC")))
                         else (if (OR (EO (@ $OC1 :OirName)
                                          (OUOTE F))
                                      (EQ (@ $OC1 :OirName)
                                          (QUOTE f)))
                                  then (if (FLOPPV.CAN.REAOP)
                                           then (TEOIT (PACKFILENAME
                                                        (QUOTE BOOY)
                                                        (CONCAT "{FLOPPY}"
                                                                (@ $M1 :Pgm)
                                                                ".OOC")))
                                         else (DODERR]
                       (BITBLT BM2 NIL NIL OWINOOW 400 400 60 60)))
                                          (* The function OMENU.WHENHELO displays a description
                                           of what function aach menu item will perform.)
```

```
(OEFINEO (OMENU.WHENHELO (ITEM.SELECTEO MENU.FROM BUTTON.PRESSEO)
                                              (* edited: " 2-Apr-B7 15:42")
                         (SELECTO ITEM.SELECTEO
                                  (OUIT (PROMPTPRINT
                                              "EXIT FROM THE OOCUMENTATION WINOOW"))
                                  (OOCUMENTATION-BY-OATE (PROMPTPRINT
                                    "OISPLAYS OOCUMENATION MAOE ON A SPECIFIC OATE"))
                                  (OOCUMENTATION-BY-PURPQSE (PROMPTPRINT
                                      "OISPLAYS OOCUMENTATION BASEO ON ITS PURPOSE"))
                                  (OOCUMENTATION-NOTES (PROMPTPRINT
                                "DISPLAYS OOCUMENTATION NOTES FOR SELECTEO PROGRAM"))
                                  (OOCUMENTATION-MANUAL (PROMPTPRINT
                                        "OISPLAYS MANUALS FOR THE SELECTEO PROGRAM"))
                                  (ENTER-OOCUMENTATION (PROMPTPRINT
                                 "ALLOWS USER TO ENTER OOCUMENTATION FOR A PROGRAM"))
                                  (PROGN NIL))))
                                              (* The function OMENU.WHENSELECTEO defines which
                                               function described ebove, to invoke when the user
                                               selects one of the options from the documentation
                                               menu.)
(OEFINEQ (OMENU.WHENSELECTEO (ITEM.SELECTEO MENU.FROM BUTTON.PRESSEO)
                           (SELECTQ ITEM.SELECTEO
                                    (OUIT (QUIT.OMENU))
                                    (OOCUMENTATION-BY-OATE (OOC.OATE))
                                    (OOCUMENTATION-BY-PURPOSE (OOC.PUROP))
                                    (OOCUMENTATION-NOTES (OOC.NOTES))
                                    (OOCUMENTATION-MANUAL (OOC.MANUALS))
                                    (ENTER-OOCUMENTATION (ENTER.OOC))
                                    (PROGN NIL))))
```

(* The function OMENUUP is used to display the documentation menu when the middle mouse button is clicked with the cursor at some point inside the documentation window. The menu is ettached to the top of the window.)

```
(OEFINEQ [OMENUUP (WINDOWNAME)
                   (if (MOUSESTATE (ONLY MIOOLE))
                       then (ATTACHWINOOW OMENU OWINOOW (OUOTE TOP])
```

(* The varieble DMENU is used to hold the record definition of the documentation window. The field title is given the value documentation window. The field items defines the ellowed options aveileble for the menu. Centerfig centers the items to be selected in the menu display. Omenu.whenheld and Omenu.whenselected determine whet messages are to be printed in the prompt window and what functions are to be called when en item is selected by clicking on it with the mouse cursor.)

```
(SETQ OMENU (MENUWINOOW (create MENU
                              |TITLE _ "OOCUMENTATION MENU"
```

```
                                          ITEMS _ (OUOTE (QUIT OOCUMENTATION-BY-OATE
                                                                OOCUMENTATION-BY-PURPOSE
                                                                OOCUMENTATION-NQTES
                                                                OOCUMENTATION-MANUAL
                                                                ENTER-OOCUMENTATION))
                                          CENTERFLG _ T
                                          WHENHELOFN _ (FUNCTION OMENU.WHENHELO)
                                          WHENSELECTEOFN _ (FUNCTION OMENU.WHENSELECTEO))
                                  T))
                                                   (* The function PRTMSG is used to prompt the user to
                                                   display the menu when no errors are found in disk
                                                   drive selection.)
(OEFINEO (PRTMSG NIL (PRINTOUT OWINOOW .SKIP 1 "MOVE THE CURSOR TO THIS WINOOW")
              (PRINTOUT OWINOOW .SKIP 2
                  "PRESS THE MIOOLE MOUSE BUTTON TO OISPLAY MENU WITH SELECTIONS")))
(CLOSEW UWINOOW)
(SETO BAOERR NIL)
                                                 (* The variable OWINOOW is used to hold the definition
                                                 of the documentation window.)
(SETO OWINOOW (CREATEW (QUOTE (500 200 500 500))
                     "OOCUMENTATION OISPLAY WINOOW" 10))

(* Setting WINOOWPROP of OWINOOW allows the clicking of a mouse button to perform the function OMENUUP wich
displays the menu on top of the documentation window.)


(WINOOWPROP OWINOOW (QUOTE BUTTONEVENTFN)
         (FUNCTION OMENUUP))

(* The remaining code prompts the user to enter the disk drive that the documentation is on.
The selections are F for floppy disk and O for hard Oisk. If the floppy drive cannot be read then an error flag is
set and the error routine OOOERR is called. In this case messages are printed and the opening window is
redisplayed.)


(PRINTOUT OWINOOW "ENTER AN F IF YOU ARE USING THE FLOPPY OISK ORIVE")
(PRINTOUT OWINOOW .SKIP 2 "ENTER A O IF YOU ARE USING THE HARO OISK ORIVE ")
(PRINTOUT OWINOOW .SKIP 2 "ENTER OISK ORIVE SELECTION ")
(TTYOISPLAYSTREAM OWINOOW)
(_@
  :OirName
  (REAO))
[if (OR (EO (@ $OC1 :OirName)
              (QUOTE O))
          (EQ (@ $OC1 :OirName)
              (QUOTE d)))
```

```
    then (_@
              :FileName
              (PACKFILENAME (OUDTE BODY)
                            (CONCAT "<LISPFILES>AL>" (@ $M1 :Pgm)
                                    ".DDC")))
   elsa (if (DR (EQ (@ $DC1 :DirName)
                    (QUOTE F))
                (EQ (@ $DC1 :OirName)
                    (OUOTE f)))
            then (if (FLOPPY.CAN.READP)
                     then (_@
                           :FileName
                           (PACKFILENAME (QUOTE BOOY)
                                         (CONCAT "(FLDPPY)" (@ $M1 :Pgm)
                                                 ".OOC")))
                   alse (DODERR]
(TTYOISPLAYSTREAM TTY)
(if (EQ BADERR T)
    then (TTYDISPLAYSTREAM TTY)
  alsa (PRTMSG))))
)

(OSK)<LISPFILES>AL>DISPLAYDOCUMENTATION.;1   8-May-87 10:46:2D  Paga: 5

(DECLARE: OONTCDPY
  (FILEMAP (NIL (368 11883 (Documentation.DisplayOocumantation 378 . 11881)))))
STDP
```

```
{OSK}<LISPFILES>AL>HISTORYOFCHANGES.;1   8-May-87 10:50:23   Page: 1


(PRETTYCOMPRINT HISTORYOFCHANGESCOMS)

(RPAQQ HISTORYOFCHANGESCOMS ((CLASSES HistoryOfChanges)
                             (METHODS HistoryOfChanges.EnterHistory HistoryOfChanges.LookForChenges)
                             ))
(DEFCLASSES HistoryOfChanges)
[DEFCLASS HistoryOfChanges
   (MetaClass Class Edited:                                  (* edited: "31-Mar-87 14:20"))
   (Supers Maintenance)
   (InstanceVariables (DirName NIL doc                       (* IV added by))
                      (FileName NIL doc                      (* IV added by))
                      (WhatToFind NIL doc                    (* IV added by))))]

[METH HistoryOfChenges  EnterHistory NIL
      (* New method template)]


[METH HistoryOfChenges  LookForChanges (WhatToLookFor FileToSearch)
      (* New method template)]


(DEFINEQ

(HistoryOfChanges.EnterHistory
  (Method ((HistoryOfChanges EnterHistory)
          self)                                              (* eln " 8-May-87 10:26")
                                                             (* New method template)

(* The method EnterHistory ellows the user to enter a history of chenges or to append new history of changas to the
existing history file for the progrem being edited. The method asks if a history file exists for the program being
maintained end ects accordingly based on the user response.)


(* The function INPUTHST allows the user to be eble to enter a history of changes for the progrem being mainteined.
This function prompts the user on how to end a sentence or history block end then calls the function OMT which
sterts the input routine.)


(DEFINEQ (INPUTHST (FLIN)
                   (SETQ HSTBLOCK FLIN)
                   (PRINTOUT HWINDOW .SKIP 2 "ENTER THE NEW HISTORY IN THE TTY WINDOW")
                   (PRINTOUT HWINDOW .SKIP 2
                          "WHEN YOU ARE DONE ENTERING A SENTENCE OR BLOCK OF CHANGES")
                   (PRINTOUT HWINDOW .SKIP 2 "PRESS THE SPACE BAR AND ENTER A . ? OR !")
```

```
                    (PRINTOUT HWINDOW .SKIP 2
                        "ONE OF THESE CHARACTERS WILL STOP THE CHANGES ENTRY FUNCTION")
                    (OMT HSTBLOCK)))

(* The function REAQHST is a recursive function that reads the information that tha user types in the tty window.
The words typed in the tty window ere CONSed into a list. In order to stop the input routina the usar must prass
the spece bar and follow with a pariod, question mark, or axclamataion point.)


(QEFINEQ [REAQHST (X)
                   (SETQ X (REAO))
                   (IF (OR (EQ X (QUOTE %.))
                               (EQ X (QUOTE !))
                               (EQ X (QUOTE ?)))
                       THEN NIL
                     ELSE (SETQ X (CONS X (REAOHST X])

(* The function OMT is a racursive function that takes as its argument the sentance or history block list thet is
to ba added to. The function CONSs new sentences onto the existing list. In order to keep the list in tha correct
ordar the input list must ba revarsed becausa CQNS attachs items to the front of e list. The function prompts the
user to sea if another sentence or history block naads to be edded to the history list. If more information is to
be added the function calls itself with the list to append to.)


(OEFINEQ (QMT (HSTBLOCK)
                 (SETQ CQMMQUT (REAQHST))
                 (SETQ HSTBLQCK (CONS CQMMQUT HSTBLOCK))
                 (PRINTOUT T "MORE? (Y/N) ")
                 (SETQ ANS (READ))
                 (IF (OR (EQ ANS (QUOTE Y))
                             (EQ ANS (QUOTE y)))
                    THEN (QMT HSTBLOCK)
                  ELSE (SETQ BIGBLOCK HSTBLQCK))))

(* Tha function DONEWHFILE is invokad when thara is no history of changas for the program thet is being maintained.
The function craetes a new file and cells the INPUTHST function so that the usar can type in history of changes
information. The printout function is used to write the history of information list to the file.
All open files ere closed end the displeystream is returned to tha tty window.)
```

```
(DEFINED (DONEWHFILE NIL (SETD FN (@ $HST1 :FileNeme))
                         (DUTFILE FN)
                         (SETO STUFFTDPRINT (INPUTHST))
                         (SETD STUFFOUT (REVERSE STUFFTOPRINT))
                         (PRINTDUT FN STUFFDUT)
                         (CLDSEALL T)
                         (TTYDISPLAYSTREAM TTY)))

(* The function DDDLOHFILE is invoked when a history of changes for the program being maintained already exists.
The function opens the old file and reads in the file information. The CAR of the readfile function is assigned to
the variable FILESTUFFIN. This is a list containing all history of changes information. This list must be reversed
because the input routine CONSs information to the front of the list. The old information is printed and the
INPUTHST function is called to allow eddition of new history information. The new information is then reversed
again and stored on a new history of changes file.)


(DEFINEQ (ODDLOHFILE NIL (SETD FN (@ $HST1 :FileName))
                         (INFILE FN)
                         (SETQ FILESTUFFIN (CAR (READFILE FN)))
                         (SETQ FSI (REVERSE FILESTUFFIN))
                         (PRINTDUT HWINOOW .SKIP 2 "THIS IS THE DLD HISTDRY OF CHANGES")
                         (PRINTDUT HWINODW .SKIP 2 FSI)
                         (CLDSEALL T)
                         (DELFILE FN)
                         (DUTFILE FN)
                         (SETQ STUFFTDPRINT (INPUTHST FSI))
                         (SETQ STUFFOUT (REVERSE STUFFTDPRINT))
                         (PRINTDUT HWINOOW .SKIP 2 "THIS IS THE NEW HISTDRY OF CHANGES")
                         (PRINTDUT HWINODW .SKIP 2 STUFFDUT)
                         (PRINTDUT FN STUFFDUT)
                         (CLDSEALL T)
                         (TTYOISPLAYSTREAM TTY)))

(* The remainder of the code prompts the user to find if the program being meinteined hes a history of changes file
or not. Oepending upon the response of the user the appropriate functions are called i.e. DDDLOHFILE or
OONEWHFILE.)


(CLEARW HWINDDW)
(PRINTDUT HWINOOW "IS THERE A HISTDRY DF CHANGES FILE FDR THIS PRDGRAM? (Y/N) ")
(TTYOISPLAYSTREAM HWINDDW)
(SETQ ANS (REAO))
```

```
(TTYDISPLAYSTREAM TTY)
(IF (OR (EO ANS (OUOTE y))
        (EO ANS (OUOTE y)))
   THEN (OOOLDHFILE)
  ELSE (DONEWHFILE))
(CLEARW HWINDOW)
(TTYDISPLAYSTREAM TTY)))


(HistoryOfChangas.LookForChenges
  (Method ((HistoryOfChanges LookForChanges)
           self WhatToLookFor FileToSearch)
                                                          (* aln "22-Apr-B7 12:21")
                                                          (* New mathod template)
(* The method LookForChanges is usad to displey the hiatory of changes in a window. Tha mathod invokes tha
recursiva function HLOOKATLIST which aearches the list for the given pattern. If the pettern is a membar of tha
list tha list is printad in the history displey window.)


(* Tha function HISMEMBEROF is used to check tha list pessed to it vie LISTTOSEARCH. Tha function checks to see if
the velue to find pessed via SEARCHVALUE is in the list given to it. If the pettern to find is in the list the
whola list is printed.)


(DEFINEO (HISMEMBEROF (LISTTOSEARCH SEARCHVALUE)
                          (SETO SV SEARCHVALUE)
                          (SETO LTS LISTTOSEARCH)
                          (if (EOMEMB SV LTS)
                              then (PRINTOUT HSTW .SKIP 2 LTS)
                            else NIL)))

(* The function HLOOKATLIST is e recursive function thet is used to saarch the history file list.
The function is passed the history file as a lerga list via LISTIN. The car of this list is than essigned to the
varieble ISIN. Tha cer should be a list also, but if it is not the list is empty end the search is completed.
If ISIN is not empty, tha word to find and ISIN is pessed to tha HISMEMBEROF function thet checks the list for the
pattern to find. The function then tekas tha COR of the hiatory list. This list will hold the remaining santencas
or blocks of history information. If this is not ampty the function then calls itself with the new list and the
seme word to find.)


(DEFINEO (HLOOKATLIST (LISTIN WORDTOFIND)
                         (SETO WTF WORDTOFIND)
                         (SETO ISIN (CAR LISTIN))
                         (if (EO ISIN NIL)
                             then NIL
                           else (HISMEMBEROF ISIN WTF))
                         (SETO LN (COR LISTIN))
```

```
                        (if (EO ISIN NIL)
                            than (PRINTOUT HSTW .SKIP 2
                                           "OONE WITH HISTORY OF CHANGES SEARCH")
                          alsa (HLOOKATLIST LN WTF))))
                                              (* The Variabla HSTW ie used to hold the definition of
                                                 tha history of chenges display window.)
(SETQ HSTW (CREATEW (OUOTE (10 10 400 400))
                    (CONCAT "HISTORY OF CHANGES FOR " (@ $M1 :Pgm))
                    5))

(* The remainder of the code takes the filaneme that was pessed to the method in FilaToSearch and assigns it to the
veriable FN. Thie file is then raed and tha Hietory list is essigned to the variable HSTLST.
The variable WTLF is assigned tha value paesed to the method vie WhatToLookFor. The function HLOOKATLIST is then
invoked with the list to eearch and whet to look for in the list.)


(CLEARW HSTW)
(SETO FN FileToSeerch)




{DSK}<LISPFILES>AL>HISTORYOFCHANGES.;1   8-May-87 10:50:23  Paga: 4

           (INFILE FN)
           (SETO HSTLST (CAR (REAOFILE FN)))
           (CLOSEALL T)
           (SETQ WTLF WhetToLookFor)
           (PRINTOUT HSTW "LOOKING FOR " WTLF)
           (HLOOKATLIST HSTLST WTLF)
           (TTYOISPLAYSTREAM TTY)))
)
(OECLARE: OONTCOPY
  (FILEMAP (NIL (1030 9832 (HistoryOfChanges.EnterHistory 1040 . 6S64) (HistoryOfChanges.LookForChanges
6S66 . 9830)))))
STOP
```

```
{DSK}<LISPFILES>AL>CHECKHISTORY.;1   8-May-87 10:31:11   Page: 1


(PRETTYCOMPRINT CHECKHISTORYCOMS)

(RPAQQ CHECKHISTORYCOMS ((METHODS HistoryOfChanges.CheckHistory)))
[METH HistoryOfChenges  CheckHistory NIL
      (* New method templete)]


(DEFINEQ

(HistoryOfChangas.CheckHistory
  (Method ((HistoryOfChanges CheckHistory)
           self)
                                                              (* eln " 8-Mey-87 10:19")
                                                              (* New method template)

(* The method CheckHistory is used to displey the history window and the manu associeted with this window.
From this function the user cen call other functions that search the history of chenges files for detes and purpose
of modules. The user cen elso display and edit all the history of changes for the program being mainteined.
Users can also enter new history of changes for the program.)



(* The function OOHERR is usad when the progrem detacts an error with the floppy disk drive.
The user is prompted to check the floppy drive and then the function OUIT.HMENU is called. This function exits to
the opening window.)


(OEFINEO (OOHERR NIL (SETO HBAOERR T)
                   (CLEARW HWINOOW)
                   (PRINTOUT HWINOOW
                  "FLOPPY NOT IN DRIVE OR OOOR NOT CLOSED.  PLEASE CHECK FLOPPY DRIVE!")
                   (PRINTOUT HWINDOW .SKIP 2
                             "THIS ERROR CAUSES A RETURN TO THE STARTUP MENU")
                   (for ZZ from 1 to 1500 do NIL)
                   (CLOSEW HWINDOW)
                   (OUIT.HMENU)))

(* The function OUIT.HMENU is used to exit from the history window end redisplay the opening window.
The window property is reset for the history window and it is then closed. The opening window is redisplayed end
tha displeystreem is sent back to the tty window.)


(OEFINEO (OUIT.HMENU NIL (WINODWPROP HWINOOW (OUDTE BUTTONEVENTFN)
                                     (FUNCTION HMENUUP))
```

```
                        (CLDSEW HWINDOW)
                        (DPENW UWINDDW)
                        (TTYDISPLAYSTREAM TTY)))
                                                 (* The function ENTER.HST is used to call the history
                                                  of changes entry function.)
(DEFINEQ (ENTER.HST NIL (_ ($ HST1)
                             EnterHistory)
                       (TTYDISPLAYSTREAM TTY)))

(* The function HST.DATE is used to display all chenges made on a specified date. The user is prompted that history
files are being loaded and then the date can be entered in the form MM-DD-19YY. The function calls the history
seerch function which displays the history of changes in a new window.)


(DEFINEQ (HST.DATE NIL (CLEARW HWINDOW)
                       (BITBLT BM1 NIL NIL HWINDOW 4DD 4DD 6D 6D)
                       (PRINTOUT HWINDDW "PLEASE WAIT!  LOADING HISTDRY OF CHANGES FILE")
                       (for ZZ from 1 to 1DDD do NIL)
                       (CLEARW HWINDOW)

                       (PRINTOUT HWINDOW "WHAT DATE WOULD YOU LIKE TD SEARCH FOR?")
                       (PRINTOUT HWINDDW .SKIP 2
                            "ENTER DATE IN FORM 'DD-MM-19YY' AND SPECIFY LEADING ZEROS")
                       (PRINTOUT HWINDOW .SKIP 2 "ENTER THE DATE TO LOOK FOR! ")
                       (TTYDISPLAYSTREAM HWINDDW)
                       (_@
                         $HST1 :WhatToFind (READ))
                       (TTYDISPLAYSTREAM TTY)
                       (_ ($ HST1)
                            LookForChenges
                            (@ $HST1 :WhatToFind)
                            (@ $HST1 :FileName))
                       (TTYDISPLAYSTREAM TTY)))

(* The function HST.PUROP is used to display all chenges dealing with a specified purpose word.
The user is prompted that history files are being loaded end then the purpose can be entered by the user.
The function calls the history search function which displeys the history of changes for that purpose in e new
window.)
```

{DSK}<LISPFILES>AL>CHECKHISTORY.;1   8-May-87 10:31:11   Page: 2

```
(DEFINED (HST.PURDP NIL (CLEARW HWINDOW)
                        (BITBLT BM1 NIL NIL HWINDOW 4DD 4DD 6D 60)
                        (PRINTDUT HWINDDW "PLEASE WAIT!  LOADING HISTORY OF CHANGES FILE")
                        (for ZZ from 1 to 1DDD do NIL)
                        (CLEARW HWINDOW)
                        (PRINTOUT HWINDOW "ENTER A WORD DESCRIBING THE PURPOSE OF A FUNCTION")
                        (PRINTOUT HWINDOW .SKIP 2 "ENTER YOUR PURPDSE WDRD! ")
                        (TTYDISPLAYSTREAM HWINDOW)
                        (_@
                          $HST1 :WhatToFind (READ))
                        (TTYDISPLAYSTREAM TTY)
                        (_ ($ HST1)
                             LookForChanges
                             (@ $HST1 :WhatToFind)
                             (@ $HST1 :FileName))
                        (TTYDISPLAYSTREAM TTY)))

(* The function HST.ALL is used to display all history of changas in a TEDIT window. The user can adit the changes
at this point if desired. Ths user is prompted that the program is loading history files.)

(DEFINEO (HST.ALL NIL (CLEARW HWINDOW)
                      (BITBLT BM1 NIL NIL HWINDOW 400 400 60 60)
                      (PRINTOUT HWINDOW "PLEASE WAIT!  LOADING HISTORY OF CHANGES FILE")
                      [if (OR (EO (@ $HST1 :DirNama)
                                      (OUOTE D))
                              (EO (@ $HST1 :DirNama)
                                      (OUOTE d)))
                         than (TEDIT (PACKFILENAME (OUOTE BODY)
                                                   (CONCAT "<LISPFILES>AL>"
                                                           (@ $M1 :Pgm)
                                                           ".HST")))
                       else (if (OR (EO (@ $HST1 :DirName)
                                            (OUOTE F))
                                    (EO (@ $HST1 :DirNama)
                                            (OUOTE f)))
                                than (if (FLOPPY.CAN.READP)
                                         then (TEDIT (PACKFILENAME
                                                        (OUOTE BODY)
                                                        (CONCAT "(FLOPPY)"
                                                                (@ $M1 :Pgm)
                                                                ".HST")))
                                       alse (DOHERR]
                     (BITBLT BM2 NIL NIL HWINDOW 400 400 60 6D)
                     (TTYDISPLAYSTREAM TTY)))
```

```
(* The function HMENU.WHENHELD is used to display what each of the options in the menu does.
A message is printed in the prompt window that describes the function of each available option.)


(DEFINEQ (HMENU.WHENHELD (ITEM.SELECTED MENU.FROM BUTTON.PRESSED)
                         (SELECTQ ITEM.SELECTED
                                  (QUIT (PROMPTPRINT "EXIT FROM THE HISTORY WINDOW"))
                                  (HISTORY-BY-DATE (PROMPTPRINT
                                               "DISPLAY CHANGES FOR A SPECIFIC DATE"))
                                  (HISTORY-BY-PURPOSE (PROMPTPRINT
                                              "DISPLAY CHANGES BY PURPOSE OF MODULE"))
                                  (HISTORY-ALL (PROMPTPRINT
                                               "DISPLAY ALL CHANGES TO THIS PROGRAM"))
                                  (ENTER-HISTORY (PROMPTPRINT
                                          "ALLOWS USER TO ENTER HISTORY OF CHANGES"))
                                  (PROGN NIL))))

(* The function HMENU.WHENHELD is used to determine which of the above functions will be executed when the user
clicks the mouse button on the desired option.)


(DEFINEQ (HMENU.WHENSELECTED (ITEM.SELECTED MENU.FROM BUTTON.PRESSED)
                             (SELECTQ ITEM.SELECTED
                                      (QUIT (QUIT.HMENU))
                                      (HISTORY-BY-DATE (HST.DATE))
                                      (HISTORY-BY-PURPOSE (HST.PUROP))
                                      (HISTORY-ALL (HST.ALL))
                                      (ENTER-HISTORY (ENTER.HST))
                                      (PROGN NIL))))

(* The function HMENUUP is used to display the history menu when the user moves the mouse cursor to the history
window and clicks the middle mouse button. The history menu is attached to the top of the history window.)


(DEFINEQ (HMENUUP (WINDOWNAME)
                  (if (MOUSESTATE (ONLY MIDDLE))
                      then (ATTACHWINDOW HMENU HWINDOW (QUOTE TOP])

(* The function HPRTMSG is used to prompt the user to move the mouse cursor to the history window and to click the
middle mouse button to display the history menu. This function is called if no errors with the disk drives occur.)


(DEFINEQ (HPRTMSG NIL (PRINTOUT HWINDOW .SKIP 1 "MOVE CURSOR TO THIS WINDOW")
                  (PRINTOUT HWINDOW .SKIP 2 "PRESS MIDDLE MOUSE BUTTON FOR MENU")))

(* The variable HMENU is used to hold the definition of the history menu. The title is given the value History Of
Changes Menu. The selection items are assigned and the Centerflg is set to true so that item selections will be
centered in the menu. The whenheld and whenselected functions are set to call the appropriate functions when the
middle or left mouse buttons are held on or clicked on a menu option.)
```

```
(SETQ HMENU (MENUWINOOW (create MENU
                                          TITLE _ "HISTORY OF CHANGES MENU"
                                          ITEMS _ (QUOTE (QUIT HISTORY-BY-OATE
                                                           HISTORY-BY-PURPOSE HISTORY-ALL
                                                           ENTER-HISTORY))
                                          CENTERFLG _ T
                                          WHENHELOFN _ (FUNCTION HMENU.WHENHELO)
                                          WHENSELECTEOFN _ (FUNCTION HMENU.WHENSELECTEO))
                                   T))
(* The remainder of the code prompts the user to enter which disk drive history of chenges reside on or will be
saved on. If there is e problem with the floppy drive, the error routine is called which will exit from the history
window.)
 (CLOSEW UWINOOW)
 (SETQ HBADERR NIL)
 (SETQ HWINOOW (CREATEW (QUOTE (500 200 500 500))
                                  "HISTORY OF CHANGES WINOOW" 10))
 (WINOOWPROP HWINOOW (QUOTE BUTTONEVENTFN)
                  (FUNCTION HMENUUP))
 (PRINTOUT HWINOOW "ENTER AN F IF YOU ARE USING THE FLOPPY ORIVE")
 (PRINTOUT HWINOOW .SKIP 2 "ENTER A O IF YOU ARE USING THE HARO OISK ORIVE")
 (PRINTOUT HWINOOW .SKIP 2 "ENTER OISK ORIVE SELECTION ")
 (TTYOISPLAYSTREAM HWINOOW)
 (_@
   :OirName
   (REAO))
 [if (OR (EQ (@ $HST1 :OirName)
                 (QUOTE O))
             (EQ (@ $HST1 :OirNeme)
                 (QUOTE d)))
     then (_@
             :FileName
             (PACKFILENAME (QUOTE BOOY)
                           (CONCAT "<LISPFILES>AL>" (@ $M1 :Pgm)
                                   ".HST")))
  else (if (OR (EQ (@ $HST1 :OirNeme)
                       (QUOTE F))
                   (EQ (@ $HST1 :OirName)
                       (OUOTE f)))
            then (if (FLOPPY.CAN.REAOP)
                     then (_@
                            :FileName
                            (PACKFILENAME (QUOTE BOOY)
                                          (CONCAT "(FLOPPY)" (@ $M1 :Pgm)
                                                  ".HST")))
                  else (OOHERR]
(TTYOISPLAYSTREAM TTY)
(if (EQ HBAOERR T)
    then (TTYOISPLAYSTREAM TTY)
  else (HPRTMSG))))
{DSK}<LISPFILES>AL>CHECKHISTORY.;1   8-May-87 10:31:11  Page: 4

)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (456 10295 (HistoryOfChanges.CheckHistory 466 . 10293)))))
STOP
```

```
{OSK}<LISPFILES>AL>EOITPROGRAM.;1  31-Mar-87 16:03:41   Page: 1


(PRETTYCOMPRINT EDITPROGRAMCOMS)

(RPAQO EDITPROGRAMCOMS ((CLASSES EditProgram)))
(OEFCLASSES EditProgram)
[OEFCLASS EditProgram
   (MetaClass Class Edited:                                (* edited: "31-Mar-87 14:21"))
   (Supers Maintenance)]

(OECLARE: OONTCOPY
  (FILEMAP (NIL)))
STOP
```

```
{DSK}<LISPFILES>AL>EDITTHESELECTEDPROGRAM.;1  22-Apr-87 11:46:23  Page: 1


(PRETTYCOMPRINT EDITTHESELECTEDPROGRAMCOMS)

(RPAQQ EDITTHESELECTEDPROGRAMCOMS ((METHODS EditProgram.EditTheSelectedProgram)))
[METH EditProgram  EditTheSelectedProgram NIL
      (* New method template)]


(DEFINEQ

(EditProgram.EditTheSelectedProgram
  (Method ((EditProgram EditTheSelectedProgram)
           self)                                             (* eln "22-Apr-87 11:27")
                                                             (* New method template)


(* The method EditTheSelectedProgram is used to allow the user to DEDIT the program being maintained.
This method displays an edit selection window and allows the user to edit the current program.
The user is prompted to move the mouse cursor to the edit window and to click the middle mouse button to display
the edit menu.)



(* The function QUIT.EMENU is used to exit from the edit window and to redisplay the opening window.
This function resets the window property associated with the edit window , closes the edit window and reopens the
opening window. It also directs the displaystream to the tty window.)


(DEFINEQ (QUIT.EMENU NIL (WINDOWPROP EWINDOW (QUOTE BUTTONEVENTFN)
                                     (FUNCTION EMENUUP))
                         (CLOSEW EWINDOW)
                         (OPENW UWINDOW)
                         (TTYDISPLAYSTREAM TTY)))

(* The function EDIT.PROG calls the method that allows the user to DEDIT the program being maintained.
A popup menu will appear when this function is called. The user should select the FNS option to edit the program.)


(DEFINEQ (EDIT.PROG NIL (_ ($ ChangesMade)
                           New
                           (QUOTE CM1))
                        (_ ($ CM1)
                           MakeChanges EWINDOW)))

(* The function EMENU.WHENHELD displays messages in the prompt window that explain what each item in the edit menu
does. The user must press and hold the left or middle mouse button to see the messages.)
```

```
(DEFINEQ (EMENU.WHENHELD (ITEM.SELECTED MENU.FROM BUTTON.PRESSED)
                              (SELECTQ ITEM.SELECTED
                                       (QUIT (PRDMPTPRINT "EXIT FROM THE EDIT WINDDW"))
                                       (EDIT-PROGRAM (PRDMPTPRINT
                                                          "EDITS SELECTED PROGRAM"))
                                       (PROGN NIL))))
                                              (* The function EMENU.WHENSELECTED will determine
                                               which of the above functions is invoked when the user
                                               clicks on one of the edit menu options.)
(OEFINEQ (EMENU.WHENSELECTED (ITEM.SELECTED MENU.FROM BUTTON.PRESSEO)
                                 (SELECTQ ITEM.SELECTEO
                                          (QUIT (QUIT.EMENU))
                                          (EDIT-PROGRAM (EDIT.PROG))
                                          (PRDGN NIL))))

(* The function EMENUUP is used to attach the edit menu to the edit window. This function is called when the user

clicks the middle mouse button while inside the edit window. The menu is attached to the top of the edit window.)


(DEFINEQ [EMENUUP (WINDDWNAME)
                (if (MOUSESTATE (DNLY MIDDLE))
                    then (ATTACHWINDDW EMENU EWINDOW (QUOTE TOP])

(* The variable EMENU is used to hold the definition of the menu record that is created. The title of the menu is
Edit Menu. The item options are set end the centerflg is assigned true. This centers the items in the menu window.
The whenheld and whenselected functions are then given the appropriate function nemes to call when the left or
middle mouse buttons ere either held down or clicked.)


(SETQ EMENU (MENUWINDOW (create MENU
                                   TITLE _ "EDIT MENU"
                                   ITEMS _ (QUOTE (QUIT EDIT-PROGRAM))
                                   CENTERFLG _ T
                                   WHENHELDFN _ (FUNCTION EMENU.WHENHELD)
                                   WHENSELECTEDFN _ (FUNCTION EMENU.WHENSELECTEO))
                            T))
(CLDSEW UWINDOW)
                                              (* The variable EWINDDW is used to hold the definition
                                               of the edit window.)
(SETQ EWINDDW (CREATEW (DUDTE (500 2D0 500 500))
                        "EDITDR STARTUP WINOOW" 1D))
                                              (* By setting the window property buttoneventfn when a
                                               mouse button is pressed while in the window it will
                                               call the function EMENUUP.)
(WINDDWPROP EWINDDW (OUOTE BUTTDNEVENTFN)
           (FUNCTION EMENUUP))
(DSK)<LISPFILES>AL>EDITTHESELECTEDPROGRAM.;1  22-Apr-87 11:46:23  Page: 2

(* The remainder of the code prompts the user to move the mouse cursor to the edit window and click the middle
mouse button to display the edit menu.)


(PRINTOUT EWINDOW "MOVE CURSOR TO THIS WINDOW")
(PRINTOUT EWINDOW .SKIP 2 "PRESS THE MIDDLE MOUSE BUTTON TO DISPLAY MENU WITH SELECTIONS")))

)
(DECLARE: DONTCOPY
  (FILEMAP (NIL (509 5044 (EditProgram.EditTheSelectedProgram 519 . 5042)))))
STOP
```

```
{DSK}<LISPFILES>AL>CHANGESMADE.;1   8-May-87 10:32:34   Page: 1


(PRETTYCDMPRINT CHANGESMADECDMS)

(RPADQ CHANGESMADECDMS ((CLASSES ChangesMade)
                        (METHDDS ChangesMade.MakeChanges)))
(DEFCLASSES ChangesMade)
[DEFCLASS ChangesMade
   (MetaClass Class Edited:                                   (* edited: "16-Apr-87 D8:28"))
   (Supers HistoryDfChanges EditProgram Documentation)]

[METH ChangesMade  MakeChanges NIL
      (* New method template)]


(DEFINEQ

(ChangesMade.MakeChanges
  (Method ((ChangesMade MakeChanges)
           self)                                              (* aln "22-Apr-87 12:DS")
                                                              (* New method template)

(* The method MakeChanges is used to DEDIT the program being maintained. The method allows the user to enter the
disk drive specification where the program resides. An F specifies the floppy drive and a D specifies the hard disk
drive. The program then checks to see if it was found and if so DEDIT for the program is invoked.
If the file is not found the program will flag the error.)

                                                  (* The function DDERR is used to display an error
                                                   message that the floppy drive is not ready.)
(DEFINED (DDERR NIL (CLEARW EWINDDW)
                    (PRINTDUT EWINDDW
                  "FLDPPY NDT IN DRIVE DR DDDR NDT CLOSED.  PLEASE CHECK FLDPPY DRIVE!")))

(* The function DDRDUTINE is called if no errors ara found with the drive specification and the program is found.
The user is prompted that the file is being loaded. The DEDIT window for the program will be displayed after the
user selects the FNS option of the popup menu that is displayed.)


(DEFINED (DDRDUTINE NIL (CLEARW EWINDDW)
                        (BITBLT BMI NIL NIL EWINDDW 40D 4D0 60 60)
                        (PRINTDUT EWINODW "PLEASE WAIT!  LDADING PRDGRAM FDR EDITING")
                        (for ZZ from 1 to 100D do NIL)
                        (CLEARW EWINDDW)
                        (LDAD FN)
                        (SETD PVAR (@ $M1 :Pgm))
                        (PRINTDUT EWINDOW "CHDDSE FNS FRDM MENU")
                        (SETQ DEditLinger NIL)
                        (DV PVAR)))
```

```
(* The remainder of the code is used to prompt the user to enter the disk drive thet the program to be maintained
resides on. If the file is found the function DOROUTINE is then invoked end the user can edit the program.)


(SETQ FN NIL)
(PRINTOUT EWINDOW .SKIP 2 "TYPE AN F IF THE PROGRAM IS ON THE FLDPPY DISK ORIVE")
(PRINTOUT EWINDOW .SKIP 2 "TYPE A D IS THE PROGRAM IS ON THE HARD DISK DRIVE")
(PRINTOUT EWINDOW .SKIP 2 "ENTER YOUR DISK ORIVE SELECTION ")
(TTYDISPLAYSTREAM EWINOOW)
(SETQ DNAME (READ))
(TTYDISPLAYSTREAM TTY)
[if (OR (EQ ONAME (OUOTE D))
           (EO ONAME (QUOTE d)))
    then [SETO FN (PACKFILENAME (QUOTE BOOY)
                                (CONCAT "<LISPFILES>AL>" (@ $M1 :Pgm]



{OSK}<LISPFILES>AL>CHANGESMADE.;1   8-May-87 10:32:34   Pege: 2

           else (if (OR (EQ DNAME (QUOTE F))
                        (EQ DNAME (QUOTE f)))
                    then (if (FLOPPY.CAN.READP)
                             then [SETO FN (PACKFILENAME (OUOTE 8ODY)
                                                         (CONCAT "(FLOPPY)"
                                                                 (@ $M1 :Pgm]
                           alsa (DOERR]
        (if (EO FN NIL)
           then (PRINTOUT EWINDOW .SKIP 2 "FILE WAS NOT FOUND")
          else (OOROUTINE))
        (TTYDISPLAYSTREAM TTY)))
)
(OECLARE: DONTCOPY
  (FILEMAP (NIL (583 3581 (ChengesMede.MakeChanges 593 . 3579)))))
STOP
```

```
{OSK}<LISPFILES>AL>PROGRAMVIEW.;1  14-Apr-87 09:56:32  Page: 1


(PRETTYCOMPRINT PROGRAMVIEWCOMS)

(RPAQQ PROGRAMVIEWCOMS ((CLASSES ProgramView)))
(DEFCLASSES ProgramView)
[DEFCLASS ProgramView
   (MetaClass Class Edited:                                  (* edited: "14-Apr-87 09:54"))
   (Supers Maintenance)]

(DECLARE: DONTCOPY
  (FILEMAP (NIL)))
STOP
```

```
(DSK)<LISPFILES>AL>VIEWOFPROGRAM.;1  22-Apr-87 11:44:58   Page: 1


(PRETTYCDMPRINT VIEWDFPRDGRAMCOMS)

(RPAOQ VIEWOFPROGRAMCOMS ((METHODS ProgramView.ViawOfProgrem)))
[METH ProgramView  ViewOfProgram NIL
      (* New method templete)]


(DEFINEQ

(ProgramView.ViewOfProgrem
  (Method ((ProgramView ViewDfProgram)
           self)                                                  (* eln "22-Apr-87 11:43")
                                                                  (* New method template)

(* The method ViewDfProgram is used to display the program view window. This allows the user to search the program
based on if the progrem wes written with a modular or a nonmoduler style. The user cen select from either moduler
or nonmoduler programming styles. When modular is aalected the user can then seerch for module names or varieble
names. When the progrem is nonmodular in construct tha user cen search only for variable names.)



(* The function QUIT.PVMENU allows the user to axit from the program viaw window and to redisplay the opaning
window. The window property buttonaventfn is reset and the progrem view window is closed. The opening window is
then reopened and the displaystream sent to the tty window.)


(OEFINEQ (QUIT.PVMENU NIL (WINDOWPROP PVWINDOW (OUOTE BUTTONEVENTFN)
                                               (FUNCTION PVMENUUP))
                          (CLOSEW PVWINDOW)
                          (OPENW UWINDDW)
                          (TTYDISPLAYSTREAM TTY)))
                                                     (* The function PV.MODULAR calls the method that
                                                      seerches e modular program.
                                                      Tha usar can seerch for module or veriable names.)
(OEFINEQ (PV.MDDULAR NIL (_ ($ Modular)
                                 New
                                 (QUOTE MD1))
                         (_ ($ MO1)
                            ViewMod)))               (* The function PV.NDNMDOULAR calls the method that
                                                      seerchas a nonmodular program.
                                                      The user cen search for veriable names only in this
                                                      routine.)
(OEFINEQ (PV.NDNMODULAR NIL (_ ($ NonModular)
                                  New
                                  (QUOTE NM1))
                            (_ ($ NM1)
                               ViewNonMod)))
```

```
(* The function PVMENU.WHENHELD ie used to displey messages in the prompt window about what each of the items in
the progrem view menu will do. When the middla or left mouse buttons ere held on a menu item a messege will appear
in the prompt window.)

(OEFINEO (PVMENU.WHENHELD (ITEM.SELECTED MENU.FROM BUTTON.PRESSED)
                           (SELECTO ITEM.SELECTED
                                    (OUIT (PROMPTPRINT
                                                  "EXIT FROM PROGRAM VIEW WINDOW"))
                                    (MODULAR-PROGRAM (PROMPTPRINT
                                                "PROGRAM IS MODULAR IN CONSTRUCT"))
                                    (NON-MODULAR-PROGRAM (PROMPTPRINT
                                            "PROGRAM IS NONMODULAR IN CONSTRUCT"))
                                    (PROGN NIL))))
```

```
(DSK)<LISPFILES>AL>VIEWOFPROGRAM.;1  22-Apr-87 11:44:58  Page: 2
```

```
(* The function PVMENU.WHENSELECTED is used to detarmina which of the ebove functions is invoked when the laft or
middla mouse buttons is clicked on a menu option.)

(DEFINEQ (PVMENU.WHENSELECTED (ITEM.SELECTED MENU.FROM BUTTON.PRESSED)
                               (SELECTO ITEM.SELECTED
                                        (OUIT (OUIT.PVMENU))
                                        (MODULAR-PROGRAM (PV.MODULAR))
                                        (NON-MODULAR-PROGRAM (PV.NONMODULAR))
                                        (PROGN NIL))))

(* The function PVMENUUP is used to attech the program view menu to the program viaw window.
The menu is attached to the top of the window.)

(DEFINEO [PVMENUUP (WINDOWNAME)
                   (if (MOUSESTATE (ONLV MIDDLE))
                       then (ATTACHWINDOW PVMENU PVWINDOW (OUOTE TOP])

(* The variable PVMENU is used to hold the definition of the menu that is craated. Tha menu title is Program View
Menu. The items to be selected are defined and the centerflg tells lisp to display the items centered in the menu.
The whenheld and whenselected functions are then definad calling the appropriate functions when the left or middle
mouse button is pressed and held or clicked on a menu item.)
(SETO PVMENU (MENUWINOOW (create MENU
                                        TITLE _ "PROGRAM VIEW MENU"
                                        ITEMS _ (OUOTE (OUIT MOOULAR-PROGRAM
                                                             NON-MOOULAR-PROGRAM))
                                        CENTERFLG _ T
                                        WHENHELOFN _ (FUNCTION PVMENU.WHENHELO)
                                        WHENSELECTEOFN _ (FUNCTION PVMENU.WHENSELECTEO))
                                T))
(CLOSEW UWINOOW)
                                              (* The veriable PVWINOOW is used to hold the
                                                 definition of the progrem view window.)
(SETQ PVWINOOW (CREATEW (OUOTE (500 200 500 500))
                                "PROGRAM VIEW WINOOW" 10))
                                               (* By setting the WINOOWPROP of the program view
                                                window the user cen display the program view menu by
                                                calling the function PVMENUUP.)
(WINOOWPROP PVWINOOW (OUOTE BUTTONEVENTFN)
                (FUNCTION PVMENUUP))

(* The remainder of the code prompts the user to move the mouse cursor to the program view window and to click the
middle mouse button to displey the program view menu.)


(PRINTOUT PVWINOOW "MOVE CURSOR TO THIS WINOOW")
(PRINTOUT PVWINOOW .SKIP 2 "PRESS THE MIOOLE MOUSE BUTTON TO OISPLAY MENU WITH SELECTIONS"))

)
)
(OECLARE: OONTCOPY
  (FILEMAP (NIL (437 5611 (ProgramView.ViewOfProgrem 447 . 5609)))))
STOP
```

```
(DSK)<LISPFILES>AL>MOOULAR.;1  23-Apr-87 10:22:39  Paga: 1


(PRETTYCOMPRINT MOOULARCOMS)

(RPAQQ MOOULARCOMS ((CLASSES Modular)
                    (METHOOS Modular.ViawMod)))
(OEFCLASSES Moduler)
[OEFCLASS Moduler
   (MetaClass Class Edited:                              (* edited: "31-Mar-87 14:22"))
   (Supars ProgremView)]

[METH Modular  ViewMod NIL
      (* New method templata)]


(OEFINEO

(Modular.ViewMod
  (Method ((Modular ViewMod)
           salf)                                         (* editad: "23-Apr-87 09:11")
                                                         (* Naw mathod templata)

(* The method ViawMod is used to display the modular viaw window. The window has tha modular view menu attached to
it. The menu allows the user to salact a module name search or e variable nama search. These two selections cell
methods thet display what thay find graphicelly. The quit option of the manu causas tha modular view search window
to close and the program view window to raopen.)



(* The function OUIT.SEARCHMENU is used to raset the modular search window, close it and reopen the program viaw
window. The display streem is returned to the tty window as wall.)


(OEFINEO (OUIT.SEARCHMENU NIL (WINOOWPROP SWINOOW (QUOTE BUTTONEVENTFN)
                                          (FUNCTION SRMENUUP))
                              (CLOSEW SWINOOW)
                              (CLOSEW MW)
                              (OPENW PVWINOOW)
                              (TTYOISPLAYSTREAM TTY)))
                                           (* The function SEARCH.MOOS is used to call the method
                                              that will display module nemes in graph form.)
(OEFINEO (SEARCH.MOOS NIL (_ ($ ModulaNames)
                                 New
                                 (QUOTE MN1))
                          (_ ($ MN1)
                              SearchForModulas)))
```

```
(* The function SEARCH.VARS is used to call the method that will display veriebles used by the program in greph
form. The window thet the instructions are to be displayed in is also passed to the search method.)

(DEFINEQ (SEARCH.VARS NIL (_ ($ VeriableNames)
                             New
                             (QUOTE VN1))
                          (_ ($ VN1)
                             SearchForVariables SWINDOW)))

(* The function SRMENU.WHENHELD is used to display messages in the prompt window about each of the menu items.
The messege is displayed after the left or middle mouse button has been pressed end held while the mouse cursor is
pointing to a menu item.)

(OEFINEQ (SRMENU.WHENHELD (ITEM.SELECTED MENU.FROM BUTTON.PRESSED)
                          (SELECTQ ITEM.SELECTED


{DSK}<LISPFILES>AL>MODULAR.;1  23-Apr-87 10:22:39  Page: 2

                                   (QUIT (PROMPTPRINT "EXIT FRDM THE SEARCH WINOOW")
                                         )
                                   (SEARCH-FOR-MODULES (PRDMPTPRINT
                                         "SEARCHES FDR SPECIFIED MOOULE NAMES"))
                                   (SEARCH-FOR-VARIABLES (PROMPTPRINT
                                         "SEARCHES FOR SPECIFIEO VARIABLE NAMES"))
                                   (PROGN NIL))))

(* The function SRMENU.WHENSELECTEO is used to determine which of the above functions will be invoked when the user
clicks the left or middle mouse button on one of the moduler search menu items.)

(DEFINEQ (SRMENU.WHENSELECTEO (ITEM.SELECTED MENU.FRDM BUTTON.PRESSED)
                              (SELECTQ ITEM.SELECTEO
                                     (QUIT (QUIT.SEARCHMENU))
                                     (SEARCH-FOR-MODULES (SEARCH.MDDS))
                                     (SEARCH-FOR-VARIABLES (SEARCH.VARS))
                                     (PROGN NIL))))

(* The function SRMENUUP is the function that is called when the user moves the mouse cursor to the modular search
window and clicks the middle mouse button. When this occurs the modular search menu is attached to the top of the
modular search window.)
```

```
(OEFINEQ [SRMENUUP (WINOOWNAME)
                        (if (MOUSESTATE (ONLY MIOOLE))
                            then (ATTACHWINOOW SRMENU SWINOOW (OUOTE TOP])

(* The verieble SRMENU is used to hold the definition of the modular search menu. The title is set to Search Menu
and the items to appeer in the menu ere determined. The centerflg is used to center the items in the menu when they
ere displeyed. The Whenheld end Whenselected functions call the eppropriata functions when the left or middle mouse
buttons ere held or clicked on a menu item.)


(SETO SRMENU (MENUWINDOW (creete MENU
                                        TITLE _ "SEARCH MENU"
                                        ITEMS _ (QUOTE (QUIT SEARCH-FOR-MOOULES
                                                             SEARCH-FOR-VARIABLES))
                                        CENTERFLG _ T
                                        WHENHELOFN _ (FUNCTION SRMENU.WHENHELO)
                                        WHENSELECTEOFN _ (FUNCTION SRMENU.WHENSELECTEO))
                               T))
(CLOSEW PVWINOOW)
                                              (* The variable SWINOOW is used to hold the definition
                                                 of the modular search window.)
(SETO SWINOOW (CREATEW (OUOTE (500 200 500 500))
                        "ITEM SEARCH WINOOW" 10))

(* By setting the window property buttoneventfn when tha middle mouse button is pressed after the mouse cursor is
inside the moduler seerch window the modular seerch menu will be displayed by the function SRMENUUP)


(WINOOWPROP SWINOOW (OUOTE BUTTONEVENTFN)
               (FUNCTION SRMENUUP))

(* The remeining code prompts the user to move the mouse cursor to the modular search window and to press the
middle mouse button to display the modular search menu.)


(PRINTOUT SWINOOW "MOVE CURSOR TO THIS WINOOW")
(PRINTOUT SWINOOW .SKIP 2 "PRESS THE MIOOLE MOUSE BUTTON TO OISPLAY MENU WITH SELECTIONS")))
)
(OECLARE: OONTCOPY
  (FILEMAP (NIL (564 5733 (Moduler.ViewMod 574 . 5731)))))
STOP
```

```
(DSK)<LISPFILES>AL>NONMODULAR.;1  23-Apr-87 10:21:56  Paga: 1


(PRETTYCOMPRINT NONMOOULARCOMS)

(RPAOO NONMOOULARCOMS ((CLASSES NonModular)
                        (METHODS NonModular.ViewNonMod)))
(OEFCLASSES NonModuler)
[OEFCLASS NonModular
   (MeteCless Class Edited:                                   (* edited: "31-Mer-87 14:22"))
   (Supers ProgremView)]

[METH NonModular  ViewNonMod NIL
      (* New method templete)]


(DEFINEO

(NonModular.ViawNonMod
  (Method ((NonModular ViewNonMod)
           self)                                              (* edited: "23-Apr-87 09:25")
                                                              (* New method template)

(* The method ViewNonMod is used to displey the non modular search window and the attached non modular search menu.
A user can search for varieble names in the progrem being maintained by using this mathod.)

                                                   (* The function OUIT.NONMOOSEARCH is used to reset the
                                                    non modular window, closa it end raopen the program
                                                    view window.)
(DEFINEO (OUIT.NONMODSEARCHMENU NIL (WINOOWPROP NMSWINDOW (OUOTE BUTTONEVENTFN)
                                                         (FUNCTION NMSRMENUUP))
                                    (CLOSEW NMSWINOOW)
                                    (CLOSEW VW)
                                    (OPENW PVWINDOW)
                                    (TTYOISPLAYSTREAM TTY)))

(* The function NMSEARCH.VARS is used to call the method thet will search for variable names in the progrem being
meintianed. The window to display messages in is elso pessed to the method.)


(OEFINEO (NMSEARCH.VARS NIL (_ ($ VeriebleNemes)
                               New
                               (OUOTE VN1))
                            (_ ($ VN1)
                               SearchForVariables NMSWINDOW)))
```

```
(* The function NMSRMENU.WHENHELD is used to display messages in the prompt window about what each of the menu
items will do when the left or middle mouse button is clicked on them.)

(DEFINEQ (NMSRMENU.WHENHELD (ITEM.SELECTEO MENU.FROM BUTTON.PRESSEO)
                            (SELECTQ ITEM.SELECTEO
                                     (QUIT (PROMPTPRINT
                                                        "EXIT FROM THE SEARCH WINODW"))
                                     (SEARCH-FOR-VARIABLES (PROMPTPRINT
                                              "SEARCHES FOR SPECIFIEO VARIABLE NAMES"))
                                     (PROGN NIL))))

(* The function NMSRMENU.WHENSELECTEO is used to determina which of the functions described above will be invoked
when a user clicks on one of the non modular menu items.)

(DEFINEQ (NMSRMENU.WHENSELECTED (ITEM.SELECTEO MENU.FROM BUTTON.PRESSEO)
                                (SELECTQ ITEM.SELECTEO
                                         (QUIT (QUIT.NONMODSEARCHMENU))

                                         (SEARCH-FOR-VARIABLES (NMSEARCH.VARS))
                                         (PROGN NIL))))

(* The function NMSRMENUUP is used to attach the non modular search menu to the non modular search window.
The menu is attached to the top of the window.)

(OEFINEQ [NMSRMENUUP (WINODWNAME)
                     (if (MOUSESTATE (ONLY MIOOLE))
                         than (ATTACHWINOOW NMSRMENU NMSWINOOW (QUOTE TOP])
(SETQ NMSRMENU (MENUWINOOW (create MENU
                                   TITLE _ "NON MOOULAR SEARCH MENU"
                                   ITEMS _ (OUOTE (QUIT SEARCH-FOR-VARIABLES))
                                   CENTERFLG _ T
                                   WHENHELDFN _ (FUNCTION NMSRMENU.WHENHELD)
                                   WHENSELECTEOFN _ (FUNCTION NMSRMENU.WHENSELECTEO)
                                   )
                           T))

(* The veriable NMSRMENU is used to hold the definition of the non modular search menu. The title is assigned the
value Non Modular Search Menu and the menu items are determined. The centerflg is used to center ell menu items
when the menu is displayed. The whenheld and whenselected functions are used to call the appropriate functions when
the left or middle mouse buttons are pressed and held or clicked upon a menu item.)

(CLOSEW PVWINOOW)                                 (* The varieble NMSWINOOW is used to hold the
                                                     definition of the non modular search window.)
(SETQ NMSWINODW (CREATEW (QUOTE (500 200 500 500))
                         "NON MOOULAR ITEM SEARCH WINOOW" 10))
(* By satting the window proparty buttoneventfn whan the usar moves the mouse to the non modular search window and
clicks the middle mouse button the function to displey the non modular search menu will be called and the menu will
ba displayed.)


(WINDOWPROP NMSWINDOW (OUDTE BUTTONEVENTFN)
               (FUNCTION NMSRMENUUP))

(* The remaining code is used to prompt the user to move the mousa cursor to the non modular search window and the
click the middle mousa button to display tha non modular search menu.)


(PRINTOUT NMSWINDDW "MOVE CURSOR TO THIS WINDOW")
(PRINTOUT NMSWINDDW .SKIP 2 "PRESS THE MIDOLE MDUSE BUTTON TO DISPLAY MENU WITH SELECTIONS")
))
)
(DECLARE: OONTCDPY
  (FILEMAP (NIL (605 5D79 (NonModular.ViewNonMod 615 . 5077)))))
STDP
```

```
{DSK}<LISPFILES>AL>MDOULENAMES.;1   8-May-87 10:42:19   Page: 1


(PRETTYCOMPRINT MDDULENAMESCDMS)

(RPAQQ MDDULENAMESCOMS ((CLASSES ModuleNemes)
                         (METHODS ModuleNames.SearchForModules)))
(OEFCLASSES ModuleNemes)
[DEFCLASS ModuleNames
   (MetaCless Class Edited:                              (* edited: "31-Mar-87 14:25"))
   (Supers ChengesMade Moduler)]

[METH ModuleNemes  SearchForModules NIL
      (* New method templete)]


(DEFINEQ

(ModulaNames.SeerchForModules
  (Method ((ModuleNemes SeerchForModules)
           self)                                         (* eln " 8-May-87 10:3D")
                                                         (* New method template)

(* The mathod SeerchForModules is used to saarch for module nemes cellsd by a program or a subprogram.
Whan module nemes are found they are displayed in greph form with tha calling function as the root and the
subfunctions es leaves in the greph tree. The user can then use the left and middle mouse buttons to look at leef
nodas of the original greph. By selecting e node with the left mouse button a naw graph is displayed showing the
functions called by that function if any. By selecting e leef node with the middle mouse button e list of variebles
if any is displayed in graph form. The user cen then eelact leef nodes from tha variables used graph.
The left mouse button will show e history of chenges for the verieble end the middla mouss button will show the
documentation that deels with thet node.)


(* The function LMBFN is used to display e greph thet shows what subfunctions e function selectad from the originel
graph cells if any. If the function calls no other functions an sppropriate message ie displayed in the search
window. The incoming node and incoming window names ere passed to this function when the left mouse button is
pressed in the originel module neme greph. Masterscopa is used to see what functione if any the incoming node neme
calls.)


(OEFINEQ [LMBFN (INCOMINGNODE INCDMINGWINDDW)
                (SETQ NDDEINTO INCOMINGNODE)
                (SETQ WINDOINTD INCDMINGWINOOW)
                (SETQ NDLABL (fatch NQDELABEL of NOOEINTD))
                [SETQ ANS (MASTERSCDPE (LIST (QUOTE WHO)
                                             (QUOTE DOES)
                                             NDLABL
                                             (QUOTE CALL]
                                    (SETQ NL NOLABL)
                                    (_ ($ HistoryOfChanges)
                                         New
                                         (QUOTE HST1))
                                    (_ ($ HST1)
                                         LookForChanges NL FN)))
                        (SETQ NODEINTO INCOMINGNODE)
                        (SETQ WINDOINTO INCOMINGWINDOW)
                        (SETQ NOLABL (fetch NODELABEL of NODEINTO))
                        (if (OR (EQ NOLABL NIL)
                                 (EQ NOLABL (QUOTE VARIABLES-USED)))
                            then (PRINTOUT SWINDOW .SKIP 2 "CHOOSE ANOTHER NODE")
                          else (BODYOFFN NOLABL))
                        (TTYDISPLAYSTREAM TTY)))

(* The function MODNMMBFN is used to display the documentation for the node selected from the vareibles used greph
or from the functions celled by graph. The function prompts the user to enter the disk drive that the documentation
files are resident on. If the file is found the method to search for the documentation for the incoming node is
called.)


(DEFINEQ (MODNMMBFN (INCOMINGNODE INCOMINGWINDOW)
                        (DEFINEQ (DOFNBODY (NOLABL)
                                    (CLEARW SWINDOW)
                                    (PRINTOUT SWINDOW
                    "TYPE AN F IF THE DOCUMENTATION FILES ARE ON FLOPPY DISK")
                                    (PRINTOUT SWINDOW .SKIP 2
                      "TYPE A D IF THE DOCUMENTATION FILES ARE ON HARD DISK")
                                    (PRINTOUT SWINDOW .SKIP 2
                                         "ENTER DISK DRIVE SELECTION ")
                                    (TTYDISPLAYSTREAM SWINDOW)
                                    (SETQ DNAME (READ))

                                    (TTYDISPLAYSTREAM TTY)
                                    [if (OR (EQ DNAME (QUOTE D))
                                              (EQ DNAME (QUOTE d)))
                                        then (SETQ FN
                                                (PACKFILENAME
                                                  (QUOTE BODY)
                                                  (CONCAT "<LISPFILES>AL>"
                                                           (@ $M1 :Pgm)
                                                           ".DOC")))
                                      else (if (OR (EQ DNAME (QUOTE F))
                                                     (EQ DNAME (QUOTE f)))
                                               then
```

```
(if (OR (EQ NDLABL NIL)
              (EQ NDLABL (@ $M1 :Pgm)))
    then (PRINTOUT SWINOOW .SKIP 2 "CHOOSE ANOTHER NODE")
  else (if (EQ ANS NIL)
              then (PRINTOUT SWINOOW .SKIP 2
                              "THIS FUNCTION CALLS NO OTHER SUBROUTINES")
            else (SHOWGRAPH (LAYOUTSEXPR (CONS NDLABL ANS)
                                          (QUOTE VERTICAL)
                                          T)
                            (CONCAT "FUNCTIONS CALLED BY " NDLABL)
                            (QUOTE LMBFN)
                            (QUOTE MMBFN)
                            T NIL NIL])

(* The function MOONLMBFN is used to display the history of changes for the node selected from either the varaibles
used graph or from the function that uses the variable graph. The function prompts the user to enter the disk drive
that the history files are resident on. If the file is found the method to search for the history of changes for
the incoming node is called.)
{DSK}<LISPFILES>AL>MODULENAMES.;1   8-May-87 10:42:19   Page: 2

(OEFINEQ (MODNLMBFN (INCOMINGNOOE INCOMINGWINOOW)
                (DEFINEQ (BODYOFFN (NDLABL)
                                (CLEARW SWINDOW)
                                (PRINTOUT SWINDOW
                         "TYPE AN F IF THE HISTORY FILES ARE ON FLOPPY DISK")
                                (PRINTOUT SWINDOW .SKIP 2
                            "TYPE A O IF THE HISTORY FILES ARE ON HARD OISK")
                                (PRINTOUT SWINDOW .SKIP 2
                                          "ENTER DISK DRIVE SELECTION ")
                                (TTYDISPLAYSTREAM SWINDOW)
                                (SETQ DNAME (READ))
                                (TTYDISPLAYSTREAM TTY)
                                [if (OR (EQ ONAME (QUOTE O))
                                              (EQ DNAME (QUOTE d)))
                                    then (SETQ FN
                                                (PACKFILENAME
                                                  (QUOTE BODY)
                                                  (CONCAT "<LISPFILES>AL>"
                                                            (@ $M1 :Pgm)
                                                            ".HST")))
                                  else (if (OR (EQ DNAME (QUOTE F))
                                                    (EQ DNAME (QUOTE f)))
                                            then
                                            (SETQ FN
                                              (PACKFILENAME
                                                (QUOTE BODY)
                                                (CONCAT "(FLOPPY)"
                                                          (@ $M1 :Pgm)
                                                          ".HST"]
                                  (SETO FN
                                   (PACKFILENAME
                                    (QUOTE BOOY)
                                    (CONCAT "(FLOPPY)"
                                          (@ $M1 :Pgm)
                                          ".OOC"]
                         (SETO NL NOLABL)
                         (_ ($ Oocumentation)
                            New
                            (OUOTE OC1))
                         (_ ($ OC1)
                            LookForOocuments NL FN)))
             (SETQ NOOEINTO INCOMINGNOOE)
             (SETO WINOOINTO INCOMINGWINOOW)
             (SETQ NOLABL (fatch NOOELABEL of NODEINTO))
             (if (OR (EQ NOLABL NIL)
                    (EQ NDLABL (OUOTE VARIABLES-USEO)))
               then (PRINTOUT SWINOOW .SKIP 2 "CHOOSE ANOTHER NOOE")
              alae (OOFNBOOY NOLABL))
             (TTYDISPLAYSTREAM TTY)))
```

(* The function MMBFN is used to displey e graph conteining the veriables used by the the node selected from the originel functions called greph. Prassing the middle mouse button will cause this graph to be displayed. The user can then select nodes of this graph to see the history or documentation about the selected node using the left and middle mouse buttons respectively.)

```
(OEFINEO [MMBFN (INCOMINGNOOE INCOMINGWINOOW)
          (OEFINEO (OOVARGRAPH (RESULT)
                        (SETO ANS RESULT)
                        (SHOWGRAPH (LAYOUTSEXPR (CONS (QUOTE

                                                  VARIABLES-USEO)
                                                    ANS)
                                                (QUOTE VERTICAL)
                                                T)
                               (CONCAT "VARIABLES USEO BY " NOLABL)
                               (OUOTE MOONLMBFN)
                               (QUOTE MOONMMBFN)
                               T NIL NIL)
                        (CLEARW SWINOOW)
                        (PRINTOUT SWINOOW
               "YOU CAN NOW USE THE MOUSE TO SELECT LEAF NOOES ON THE GRAPH")
                        (PRINTOUT SWINOOW .SKIP 2
                      "MOVE CURSOR TO A OESIREO LEAF IN THE GRAPH TREE")
                        (PRINTOUT SWINOOW .SKIP 2
          "PRESS LEFT MOUSE BUTTON TO SEE HISTORY OF CHANGES FOR SELECTEO NOOE")
                        (PRINTOUT SWINOOW .SKIP 2
            "PRESS MIODLE MOUSE BUTTON TO SEE OOCUMENTATION FOR SELECTEO NOOE")))
          (SETO NOOEINTO INCOMINGNOOE)
          (SETQ WINOOINTO INCOMINGWINOOW)
          (SETQ NOLABL (fetch NOOELABEL of NOOEINTO))
```

```
                [SETO ANS (MASTERSCOPE (LIST (QUOTE WHD)
                                                (OUOTE DOES)
                                                NDLABL
                                               .(QUOTE USE]
                (if (OR (EQ NDLABL NIL)
                        (EQ NDLABL (OUOTE VARIABLES-USEO)))
                    then (PRINTOUT SWINDOW .SKIP 2 "CHOOSE ANDTHER NODE")
                  else (if (EO ANS NIL)
                           then (PRINTOUT SWINDOW .SKIP 2
                                         "THIS SUBROUTINE USES NO LOCAL VARIABLES")
                         else (OOVARGRAPH ANS])

(* The function PRTMODPROMPTMSG is used to prompt the user that they can use the left and middle mouse buttons to
display new graphs by selecting the correct nodes.)


(DEFINEQ (PRTMODPROMPTMSG NIL (PRINTOUT SWINDOW .SKIP 2
                         "YOU CAN NDW USE THE MOUSE TO SELECT LEAF NODES ON THE GRAPH")
                           (PRINTOUT SWINDOW .SKIP 2
                                     "MOVE CURSOR TO A DESIRED LEAF IN THE GRAPH TREE")
                           (PRINTOUT SWINDOW .SKIP 2
                "PRESS LEFT MOUSE BUTTON TO SEE FUNCTIONS CALLEO BY THIS SUBROUTINE")
                           (PRINTOUT SWINDOW .SKIP 2
                "PRESS MIOOLE MOUSE BUTTON TO SEE VARIABLES USED BY THIS SUBROUTINE")))

(* This section of code uses masterscope to see what functions the main program being maintained calls.
The program must be loaded and run for masterscope to be able to analyze the program and return the correct
enswers. The user is prompted that they cen use the left or middle mouse buttons to display new graphs each with
their own node selection functions.)


(SETO MW NIL)
(CLEARW SWINDOW)
(PRINTOUT SWINDOW "SEARCHING FOR MOOULE NAMES")
(MASTERSCOPE (OUOTE (ERASE)))
(SETO OUES (LIST (OUDTE ANALYZE)
                 (@ $M1 :Pgm)))
(MASTERSCOPE DUES)
(SETO OUES (LIST (DUOTE WHO)
                 (DUOTE DOES)
                 (@ $M1 :Pgm)
                 (OUOTE CALL)))
(SETO ANSBACK (MASTERSCOPE QUES))
(if (EQ ANSBACK NIL)
    then (PRINTOUT SWINDOW .SKIP 2 "THIS PROGRAM CALLS NO FUNCTIDNS")
  else (SETD MW (SHOWGRAPH (LAYOUTSEXPR (CONS (@ $M1 :Pgm)
(DSK)<LISPFILES>AL>MODULENAMES.;1   8-May-87 10:42:19  Page: 4
                                                                  ANSBACK)
                                                        (OUOTE VERTICAL)
                                                        T)
                                    (CONCAT "FUNCTIONS CALLEO BY " (@ $M1 :Pgm))
                                    (OUOTE LMBFN)
                                    (OUOTE MMBFN)
                                    T NIL NIL)))
(IF (EO ANSBACK NIL)
    THEN NIL
  ELSE (PRTMOOPROMPTMSG))
(TTYOISPLAYSTREAM TTY)))
)
(OECLARE: OONTCOPY
  (FILEMAP (NIL (627 11102 (ModuleNames.SearchForModules 637 . 11100)))))
STOP
```

```
{DSK}<LISPFILES>AL>VARIABLENAMES.;1    8-May-87 10:38:26   Paga: 1


(PRETTYCOMPRINT VARIABLENAMESCOMS)

(RPAQQ VARIABLENAMESCOMS ((CLASSES VariableNamas)
                          (METHODS VariableNames.SearchForVariablas)))
(DEFCLASSES VariebleNames)
[DEFCLASS VariablaNamea
   (MetaClass Class Edited:                                     (* editad: "31-Mar-87 14:25"))
   (Supars ChengasMade Modular NonModular))

[METH VariableNames  SearchForVariables (WindowToDiaplayIn)
      (* New mathod tamplate)]


(DEFINEO

(VariableNames.SaarchForVariablas
  (Method ((VeriablaNames SaarchForVeriables)
           self WindowToDispleyIn)                             (* aln " 8-May-87 10:27")
                                                               (* Naw method template)

(* Tha method SearchForVariebles is used to displey in greph form all variables used by the program baing
maintainad. The method usas mastarscope to datermina if any variebles ara used by tha program and than displays
them in a graph. The user can then selact nodas on tha graph producad to find who cells the variable.
This graph can than be saarchad by pressing the left or middle mouse buttons. The laft mouse button will show a
history of changes for the noda passed to it and the middle mouse button will show tha documentation associated
with that noda.)


(* The function LMB is used than the left mouse button is prassed in tha original variables used by program graph.
This functions craatas and displays a naw greph with the incoming noda passad to it. The function then prompts the
usar that the left and middla mouse buttons can be used to display a history or documentation for tha node selacted
from the graph this function displays.)


(DEFINEO [LMB (INCOMINGNODE INCOMINGWINDOW)
                (DEFINEO (SHOWVARGRAPH (RESULT)
                                  (SETO ANS RESULT)
                                  (SHDWGRAPH (LAYOUTSEXPR (CDNS (OUDTE

                                                       FUNCTION-THAT-USES-VARIABLE)
                                                                ANS)
                                                           (QUOTE VERTICAL)
                                                           T)
                                                   (CONCAT "VARIABLE " NDLABL
                                                           " IS USED BY")
```

```
                                                          (QUOTE NLMBFN)
                                                          (QUOTE NMMBFN)
                                                          T NIL NIL)
                                            (CLEARW OISPLAYWIND)
                                            (PRINTOUT OISPLAYWIND
                                              "YOU CAN NOW USE THE MDUSE IN THE NEW GRAPH")
                                            (PRINTOUT OISPLAYWIND .SKIP 2
                           "PRESS LEFT MDUSE BUTTON FOR HISTDRY DF CHANGES FOR SELECTION")
                                            (PRINTOUT DISPLAYWIND .SKIP 2
                              "PRESS MIODLE MOUSE BUTTON FDR DOCUMENTATIDN FOR SELECTION")))
                  (SETQ NODEINTD INCOMINGNOOE)
                  (SETQ WINDDINTO INCOMINGWINDOW)
                  (SETQ NDLABL (fetch NDDELABEL of NDOEINTO))
                  (SETQ ANS (MASTERSCOPE (LIST (QUOTE WHO)
                                                      (QUOTE USES)
                                                      NDLABL)))

                  (if (OR (EQ NDLABL NIL)
                              (EQ NOLABL (@ $M1 :Pgm))
                              (EQ NDLABL (QUDTE VARIABLES-USEO))
                              (EQ NDLABL (QUOTE FUNCTION-THAT-USES-VARIABLE)))
                      then (PRINTOUT OISPLAYWIND .SKIP 2 "CHOOSE ANDTHER NODE")
                    else (if (EQ ANS NIL)
                              then (PRINTDUT DISPLAYWIND .SKIP 2
                                                "NDDE SELECTED WAS NDT A VARIABLE")
                            else (SHOWVARGRAPH ANS])

(* The function NLMBFN is used to check for a history of changes for the node selected from the graph created by
the LBM function. The user is prompted to enter the disk drive that the history files are resident on.
If the files are found the method that searches for a history of changes is called and the item to be looked for is
passed to it.)

{OSK}<LISPFILES>AL>VARIABLENAMES.;1   B-May-B7 10:3B:26   Page: 2

(DEFINEO (NLMBFN (INCOMINGNODE INCOMINGWINDDW)
                    (DEFINEQ (DOTHEFN (NDLABL)
                                         (CLEARW OISPLAYWIND)
                                         (PRINTDUT OISPLAYWIND
                                      "TYPE AN F IF THE HISTDRY FILES ARE ON FLOPPY DISK")
                                         (PRINTOUT OISPLAYWIND .SKIP 2
                                         "TYPE A D IF THE HISTORY FILES ARE ON HARO DISK")
                                         (PRINTOUT OISPLAYWIND .SKIP 2
                                                    "ENTER DISK DRIVE SELECTION ")
                                         (TTYOISPLAYSTREAM DISPLAYWIND)
                                         (SETO ONAME (READ))
                                         (TTYOISPLAYSTREAM TTY)
                                         [if (DR (EQ DNAME (QUOTE D))
                                                     (EO ONAME (QUDTE d)))
                                             then (SETQ FN
                                                      (PACKFILENAME (QUOTE BODY)
                                                            (CDNCAT
                                                               "<LISPFILES>AL>"
                                                                      (@ $M1 :Pgm)
                                                                      ".HST")))
                                    else (if (OR (EQ ONAME (QUDTE F))
                                                     (EQ DNAME (QUOTE f)))
                                             then (SETQ FN
                                                     (PACKFILENAME
                                                       (QUOTE BOOY)
                                                       (CONCAT "{FLOPPY}"
                                                                 (@ $M1 :Pgm)
                                                                 ".HST"]
                                   (SETQ NL NDLABL)
                                   (_ ($ HistoryOfChenges)
                                       New
                                       (QUDTE HST1))
                                   (_ ($ HST1)
                                       LookForChenges NL FN)))
                      (SETQ NDDEINTO INCDMINGNODE)
                      (SETQ WINDOWINTO INCDMINGWINDOW)
                      (SETO NDLABL (fetch NOOELABEL of NDDEINTO))
                      (if (DR (EQ NDLABL NIL)
                              (EQ NOLABL (QUOTE VARIABLES-USED))
                              (EQ NDLABL (QUOTE FUNCTION-THAT-USES-VARIABLE)))
                         then (PRINTDUT OISPLAYWIND .SKIP 2 "CHOOSE ANDTHER NODE")
                        else (OOTHEFN NDLABL))
                      (TTYDISPLAYSTREAM TTY)))
```

(* The function NMMBFN is invoked when a user presses the middle mouse button in the greph created by the function LMB. This function looks for the documentation associated with the node that wes passed to it.
The function calls the method that searches for the documentetion end passes it whet to look for in the documentation files. The user is prompted to enter the disk drive that the documentation files are resident on.
If the file is found the documentation for the node is displeyed in a new window.)

```
(DSK)<LISPFILES>AL>VARIABLENAMES.;1   8-May-87 10:38:26  Page: 3

    (DEFINEQ (NMMBFN (INCDMINGNODE INCDMINGWINOOW)
                   (OEFINEQ (OOTHEFNBOOY (NDLABL)
                                         (CLEARW DISPLAYWIND)
                                         (PRINTDUT DISPLAYWIND
                            "TYPE AN F IF THE ODCUMENTATION FILES ARE ON FLOPPY DISK")
                                         (PRINTOUT DISPLAYWIND .SKIP 2
                              "TYPE A D IF THE DOCUMENTATION FILES ARE ON HARD DISK")
                                         (PRINTOUT DISPLAYWIND .SKIP 2
                                                  "ENTER DISK ORIVE SELECTION ")
                                         (TTYDISPLAYSTREAM OISPLAYWIND)
                                         (SETQ DNAME (REAO))
                                         (TTYDISPLAYSTREAM TTY)
                                         [if (OR (EQ ONAME (QUOTE O))
                                                        (EQ DNAME (QUOTE d)))
                                                  then (SETQ FN
                                                           (PACKFILENAME
                                                             (QUDTE BDDY)
                                                             (CDNCAT "<LISPFILES>AL>"
                                                                        (@ $M1 :Pgm)
                                                                        ".DDC")))
                                               else (if (DR (EQ DNAME (QUDTE F))
                                                                    (EQ DNAME (QUDTE f)))
                                                           then
                                                            (SETQ FN
                                                              (PACKFILENAME
                                                                (QUDTE BDDY)
                                                                (CDNCAT "(FLOPPY)"
                                                                          (@ $M1 :Pgm)
                                                                          ".DDC"]
                                             (SETQ NL NDLABL)
                                             (_ ($ Documentation)
                                                  New
                                                  (QUOTE DC1))
                                             (_ ($ DC1)
                                                  LookForDocuments NL FN)))
                    (SETQ NDDEINTD INCDMINGNDDE)
                    (SETQ WINDDWINTD INCDMINGWINDDW)
                    (SETQ NDLABL (fetch NDDELABEL of NDDEINTD))
                    (if (DR (EQ NDLABL NIL)
                                 (EQ NDLABL (QUDTE VARIABLES-USED))
                                 (EQ NDLABL (QUDTE FUNCTIDN-THAT-USES-VARIABLE)))
                        then (PRINTDUT DISPLAYWIND .SKIP 2 "CHDDSE ANDTHER NDDE")
                      else (DDTHEFNBDDY NDLABL))
                    (TTYDISPLAYSTREAM TTY)))
```

(* The function PRTVARPRDMPTMSG is used to prompt the user thet by selecting a node of the original variables used graph that they cen displey a new graph showing whet functions use that varieble.)

```
(DEFINEQ (PRTVARPRDMPTMSG NIL (PRINTDUT DISPLAYWIND .SKIP 2
                           "YDU CAN NDW USE THE MDUSE TD SELECT LEAF NDDES DN THE GRAPH")
                             (PRINTDUT DISPLAYWIND .SKIP 2
                                       "MDVE CURSDR TD A DESIRED LEAF IN THE GRAPH TREE")
                             (PRINTDUT DISPLAYWIND .SKIP 2
                                 "PRESS LEFT MDUSE BUTTDN TD SEE WHD USES THIS VARIABLE")))
```

(* This section of code uses mesterscope to search for the variables used by the program being maintained.
If veriables ere found then the veriebles used graph is displeyed end the user is prompted thet the use of the left mouse button will displey a graph thet will show who uses the varieble.)

```
(SETQ VW NIL)
(SETQ DISPLAYWIND WindowToDispleyIn)
```

```
(DSK)<LISPFILES>AL>VARIABLENAMES.;1    B-May-B7 10:3B:26   Page: 4

          (CLEARW OISPLAYWINO)
          (PRINTOUT OISPLAYWINO "SEARCHING FOR VARIABLES")
          (MASTERSCOPE (OUOTE ERASE))
          (SETO OUES (LIST (OUOTE ANALYZE)
                           (@ $M1 :Pgm)))
          (MASTERSCOPE OUES)
          (SETO OUES (LIST (QUOTE WHO)
                           (QUOTE OOES)
                           (@ $M1 :Pgm)
                           (QUOTE USE)))
          (SETQ ANSBACK (MASTERSCOPE QUES))
          (if (EQ ANSBACK NIL)
              then (PRINTOUT OISPLAYWINO .SKIP 2 "THIS PROGRAM USES NO VARAIBLES")
            else (SETQ VW (SHOWGRAPH (LAYOUTSEXPR (CONS (OUOTE VARIABLES-USEO)
                                                        ANSBACK)
                                                  (OUOTE VERTICAL)
                                                  T)
                                     (CONCAT "VARIABLES USEO BY " (@ $M1 :Pgm))
                                     (OUOTE LMB)
                                     NIL T NIL NIL)))
          (IF (EO ANSBACK NIL)
              THEN NIL
            ELSE (PRTVARPROMPTMSG))
          (TTYOISPLAYSTREAM TTY)))
)
(OECLARE: OONTCOPY
  (FILEMAP (NIL (686 9704 (VariableNames.SearchForVariables 696 . 9702)))))
STOP
```

```
(PRETTYCOMPRINT BM1COMS)

(RPAQQ BM1COMS ((VARS BM1)))

(RPAQ BM1 (READBITMAP))
(6D 6D
"OOOOOOOOOOOOOOO@"
"OOOOOOOOOOOOOOO@"
"L@@@@@@@@@@@@@C@"
"MOOOOOOOOO@@@@C@"
"M@@@@@@@@A@@@@C@"
"M@@@@@@@@A@@@@C@"
"M@@@@@@@@A@@@@C@"
"MBANOGBIGI@@@@C@"
"MBABIEJMOA@@@@C@"
"MBABIOJMOA@@@@C@"
"MBABOOJKEI@@@@C@"
"MBABIDJKOI@@@@C@"
"MBABIEJIDI@@@@C@"
"MCMNIGBIGI@@@AO@"
"M@@@@@@@@A@@@AO@"
"M@@@@@@@@A@@@AO@"
"MOOOOOOOOO@@@AO@"
"L@@@@@@@@@@@@AO@"
"L@@@@@@@@@@@@@C@"
"L@@@@@@@@@@@@@C@"
"L@@@@@@@@@@@@@C@"
"L@@@@@@@@@@@@@C@"
"L@@@@@COH@@@@@C@"
"L@@@@@F@L@@@@@C@"
"L@@@@@MOF@@@@@C@"
"L@@@@AKOK@@@@@C@"
"L@@@@AGOM@@@@@C@"
"L@@@@AGOM@@@@@C@"
"L@@@@AGOM@@@@@C@"
"L@@@@AGOM@@@@@C@"
"L@@@@AGOM@@@@@C@"
"L@@@@AKOK@@@@@C@"
"L@@@@@MOF@@@@@C@"
"L@@@@@F@L@@@@@C@"
"L@@@@@COH@@@@@C@"
"L@@@@@@@@@@@@@C@"
"L@@@@@@@@@@@@@C@"
"L@@@@@@@@@@@@@C@"
"L@@@@@@N@@@@@@C@"
"L@@@@@AO@@@@@@C@"
"L@@@@@CAH@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@B@H@@@@@C@"
"L@@@@@CAH@@@@@C@"
"L@@@@LAO@F@@@@C@"
```

```
"L@@@AN@N@O@@@@C@"
"L@@@AN@@@O@@@@C@"
"OOOOOOOOOOOOOOO@"
"OOOOOOOOOOOOOOO@")
(OECLARE: OONTCOPY
  (FILEMAP (NIL)))
STOP
```

```
(DSK)<LISPFILES>AL>BM2.;1  11-Apr-87 13:03:53  Page: 1

(PRETTYCOMPRINT BM2COMS)

(RPAQQ BM2COMS ((VARS BM2)))

(RPAQ BM2 (READBITMAP))
(60 60
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"

(DSK)<LISPFILES>AL>BM2.;1  11-Apr-87 13:03:53  Page: 2

"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@"
"@@@@@@@@@@@@@@@@")
(DECLARE: DONTCOPY
  (FILEMAP (NIL)))
STOP
```

```
{DSK}<LISPFILES>AL>TEST1.;3  15-Apr-87 13:14:07  Page: 1


(PRETTYCOMPRINT TEST1COMS)

(RPAOO TEST1COMS ((FNS TEST1)))
(OEFINEO

(TEST1
  [LAMBOA NIL                                                      (* edited: "15-Apr-87 11:41")
    (PRINTOUT T "THIS IS AN EXAMPLE PROGRAM")
    (PRINTOUT T .SKIP 2 "MORE LATER")
    (OEFINEO (REAOIT NIL (PRINTOUT T .SKIP 2 "PLEASE ENTER SOME STUFF ")
                     (SETO STUFF (REAO))))
    (DEFINEO (PRINTIT NIL (PRINTOUT T .SKIP 2 "THIS IS THE STUFF THAT YOU TYPED IN " STUFF)))
    (OEFINEO (OOIT NIL (READIT)
                    (PRINTIT)))
    (DOIT)
    (SETO X 0)
    (SETO Y 0)
    (PRINTOUT T .SKIP 2 "HERE'S X " X)
    (PRINTOUT T .SKIP 2 "HERE'S Y " Y T])
)
(OECLARE: OONTCOPY
  (FILEMAP (NIL (246 867 (TEST1 256 . 865)))))
STOP
```

```
(DSK) LISPFILES AL MAINTENANCE.STARTUP.DOC.;1  8-MAY-87 11:24:34  Page: 1
((The method Maintenance.StartUp is used to display a user prompt
window) (This window has a menu
attached to it that allows the user to enter and search for
documentation, enter and search for a
history of changes, edit the selected program, and search the
program for module and variable names) (
The method prompts the user for their name and for the program to
be worked on) (The user must move
the cursor to the user prompt window and click the middle mouse
button to bring up the menu) (Once the
 menu has been displayed, to invoke the desired function, move
the cursor to the correct menu item and
 click the left or middle mouse button) (The function QUIT.STMENU
is used to exit from MAT1) (
QUIT.STMENU resets the user prompt window's buttonevent function
and closes the opening window) (
QUIT.STMENU then returns the output stream to the tty window)
(The function SEE.DOC starts the
documentation entry and display feature of MAT1) (The function
SEE.HST starts the history of changes
entry and display feature of MAT1) (The function EDIT.PGM starts
the editing feature of MAT1) (The
function SRCH.PGM starts the searching feature of MAT1) (The
function STMENU.WHENHELD displays a
description of what each item in the startup menu will do) (The
function STMENU.WHENSELECTED defines
which function to invoke when the user selects one of the options
from the startup menu) (The function
 MENUUP is used to display the startup menu when the middle mouse
button is clicked with the mouse
cursor in the user prompt window) (MENUUP uses attachwindow to
attach the menu to the top of the user
prompt window) (The variable STMENU is used to hold the
definition of the startup menu) (The variable
```

LOGO is used to store the definition of the logo window that
displays Maintenance Assistance Tool on
the bottom of the screen) (The variable UWINDOW is used to store
the definition of the user prompt
window) (Setting WINDOWPROP tells lisp that when the cursor is
moved to the window UWINDOW it will
call the function MENUUP when the middle mouse button is clicked)
(The remaining code prompts the user
to enter their name and the name of the program to be
maintained) (It also prompts the user to move
the mouse cursor to the user prompt window and to click the
middle mouse button to display the opening
menu) (The variable TTY is used to return the display stream to
the tty window) (The variable
ITEM.SELECTED is used to select an item from a menu))

```
(DSK) LISPFILES AL TEST1.HST.;1 8-MAY-87  9:45:12  Page: 1
((TEST1 IS AN EXAMPLE PROGRAM USED TO SHOW THE USEFULNESS OF
MAT1) (TEST1 HAS TWO SUBROUTINES READIT
AND PRINTIT) (THESE SUBROUTINES ARE INVOKED BY THE ROUTINE DOIT)
(READIT READS INFO ENTERED FROM THE
KEYBOARD AND PLACES IT IN THE VARIABLE STUFF) (PRINTIT PRINTS THE
INFO THAT STUFF HOLDS) (TEST1 USES
THE VARIABLES X AND Y) (TEST1 SETS X AND Y TO THE VALUE 0) (TEST1
PRINTS THE VALUES OF X AND Y AND
THEN ENDS) (TEST1 WAS CREATED ON 04-21-1987))
```

```
(DSK) LISPFILES AL MAINTENANCE.STARTUP.HST.;1  8-MAY-87 11:36:20  Page: 1
((The method Maintenance.StartUp was finalized on 22-04-1987)
(Maintenance.StartUp has evolved through
 many changes while MAT1 was being implemented) (These changes to
Maintenance.StartUp are too numerous
 to consider mentioning at this point in time) (The function
QUIT.STMENU has been altered several
times) (QUIT.STMENU was last changed on 07-04-1987) (The function
SEE.DOC was last changed on
07-04-1987) (The function SEE.HST was last changed on 07-04-1987)
(The function EDIT.PGM was last
changed on 07-04-1987) (The function SRCH.PGM was last changed on
07-04-1987) (The function
STMENU.WHENHELD has been altered several times) (The last time
that STMENU.WHENHELD was changed was on
 07-04-1987) (The function STMENU.WHENSELECTED was last changed
on 07-04-1987) (The variable STMENU
was altered on 22-04-1987) (The variable LOGO was changed on
07-04-1987) (The variable UWINDOW was
changed on 07-04-1987) (The variable TTY was changed on
07-04-1987))
```

```
 (DSK) LISPFILES AL TEST1.HST.;1  8-MAY-87 10:10:00  Page: 1
((TEST1 WAS CREATED ON 04-21-1987) (THE VARIABLE STUFF WAS
CREATED ON 04-21-1987) (STUFF HAS NOT BEEN
CHANGED) (THE VARIABLE X WAS CREATED ON 04-21-1987) (X HAS NOT
BEEN CHANGED) (THE VARIABLE Y WAS CREATED
 ON 04-21-1987) (Y HAS NOT BEEN CHANGED) (THE ROUTINE DOIT WAS
CREATED ON 04-21-1987) (DOIT HAS NOT BEEN
 CHANGED) (THE ROUTINE READIT WAS CREATED ON 04-21-1987) (THE
ROUTINE READIT HAS NOT BEEN CHANGED) (
THE ROUTINE PRINTIT WAS CREATED ON 04-21-1987) (THE ROUTINE
PRINTIT HAS NOT BEEN CHANGED))
```

A KNOWLEDGE BASED TOOL TO AID IN SOFTWARE MAINTENANCE

by

ALBERT L. NICHOL

B.S., Kansas State University, 1984

AN ABSTRACT OF A MASTER'S THESIS

Submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1987

ABSTRACT

One of the most overlooked parts of the software life cycle is the phase called maintenance. Until recently, little attention has been given to this task. The attention that has been given to this phase focused mainly on how to develop maintainable software by using better software design techniques. Attention focused on the design of software, to aid in maintenance, does not address the problems that maintainers are facing today. Maintainers need tools that will aid them while they are performing a maintenance task. Tools that will help to speed the learning portion of maintenance must be developed. Documentation and a history of changes for the program being maintained are the best sources for gaining the knowledge needed to do a maintenance task.

This thesis describes a tool that will aid in the learning portion of maintenance. The tool is called Maintenance Assistance Tool or MAT1. The tool uses a knowledge base to obtain information about the program being maintained. MAT1 was developed using the LOOPS programming environment and object-oriented programming techniques. A documentation and history of changes search and display, along with graphic representation of program parts is used to provide a learning environment in which a maintainer can learn the functions of the program being maintained quickly and easily. The program can also be edited using MAT1. The use of the LOOPS environment allows for easy and efficient expansion of MAT1 for future use.