ANTICS - A System for Animating LISP Programs

by

Mark Stephen Dionne

B.Sc.(with honor), Michigan State University, 1972

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in

THE DEPARTMENT OF COMPUTER SCIENCE

We accept this thesis as conforming
to the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

October, 1975

© Mark Stephen Dionne, 1975

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study.

I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date October 27, 1975

ABSTRACT

An animated film is an excellent medium for conveying complex ideas in computer science. A system has been developed which produces animated films, film strips, or slides depicting the execution of LISP programs. The design and implementation of this system is discussed and it is compared to existing systems.

The system, named ANTICS, may be used by entering very simple commands, and produces real-time animation. The system may be "backed up" interactively and atomic values may be changed. Advanced commands and a set of graphics primitives are available which permit an instructor or film-maker to control minute details of the animation and to add features not explicitly provided by the system. ANTICS may therefore be used as an interactive educational tool or as an animation system.

ANTICS is very economical to use. A three minute film showing the operation of the recursive function MEMBER was produced for a total cost of \$12.00. The film is included as part of the thesis. The system is dependent on IBM 370/168 and Adage Graphics Terminal hardware, but the design, which is based on the organization of the LISP EVAL function, could be used on other systems.

TABLE OF CONTENTS

CHAPTER I - PROGRAM ANIMATION	1
1.1 Introduction	1
1.2 Motivation	1
1.3 Relevant Literature	3
1.4 Types of Animation	4
1.5 Animation Methods	5
1.5.1 Key frame animation	5
1.5.2 Simple animation	6
CHAPTER II - SYSTEM DESIGN	8
2.1 Display Design	8
2.1.1 S-expressions	8
2.1.2 Evaluation	10
2.2 Single-frame vs. Real-time Animation	13
2.3 Script Design	15
2.4 Graphics Primitives	16
2.5 Human Factors Affecting Design	17
CHAPTER III - IMPLEMENTATION	21
3.1 System Organization	21
3.1.1 The Adage Graphics Terminal	21
3.1.2 The GRAPH supervisor program	22
3.1.3 LISP BASIC	22
3.1.4 LISP/MTS	23
3.2 Display File Organization	23
3.3 Methods Used in the Program	31
3.4 Cost and Execution Time	32
CHAPTER IV - CONCLUSIONS	33

4.1 Comparison With Existing Systems	33
4.2 Possible Improvements to ANTICS	33
4.3 Summary	34
BIBLIOGRAPHY	35
APPENDIX A - ANTICS USER'S GUIDE	39
1. The ANTICS Script	39
2. How to Run ANTICS	41
3. Using the System Interactively	42
4. The ANTICS EVAL Function	43
5. Additional Commands	45
6. Graphics Language	45
7. Interfacing with GLISP	49
8. Producing Films	50
9. Commands	54
10. Graphics Primitives	60
11. Useful ANTICS Variables	62
Index to ANTICS User's Guide	68
APPENDIX B - SAMPLE ANTICS SCRIPTS	69
APPENDIX C - ANTICS PROGRAM LISTING	73

Film accompanying this thesis is filed in Bookcase 2,
Special Collections

LIST OF FIGURES

1 • An Example of the Display of S-expressions	9
2 • Snapshot of EVAL Animation	12
3 • Display File Organization	26
4 • Dimensions of an EVAL Box	65
5 • Dimensions of a CONS-cell	66

ACKNOWLEDGEMENT

I would like to express my appreciation to Dr. Alan Mackworth for suggesting this thesis and for his helpful suggestions for making the system more versatile. I would also like to thank Dr. Barry Pollack and Dr. Richard Rosenberg for their suggestions for improving this thesis.

CHAPTER I - PROGRAM ANIMATION

1.1 INTRODUCTION

Animated films and displays have shown potential of becoming useful and practical teaching aids in computer science education. Animation can be used to illustrate actions which normally cannot be seen, and which are difficult to express on a blackboard because of their complexity or dynamics. These conditions often occur while teaching computer science.

A system named ANTICS has been produced which animates programs written in LISP [16]. The system is designed for use by an instructor in a programming course, allowing her to make films which will demonstrate various features of the LISP language and of algorithms programmed in LISP. The instructor may control the parts of LISP which are animated and the amount of detail which is shown. Also, the system is simple and economical enough to be used as an on-line interactive instructional aid.

1.2 MOTIVATION

Teaching beginning computer science almost always has involved graphic tools -- a quick look through many textbooks will verify this. Flowcharts, data structures, system organization diagrams, parse trees, hardware schematics and graphic representations of certain algorithms each have an

important place in the language of computer science.¹ Many computer science concepts, such as the stack, the linked list, and the array have an implicit graphic language of their own which is invariably taught in beginning courses. Other more dynamic concepts do not have well established graphic representations, since their actions are not easy to convey with fixed images. Examples of these are recursion, iteration, binding, and algorithms such as searching, sorting, and parsing.

The motive for animating computer programs is to provide new graphic expressions for complex dynamic processes in the field of computer science. (Complex ideas in physics already have been illustrated in several computer produced films [10, p. 313].) A visual display of dynamic processes should greatly enhance the student's appreciation of complex systems which are difficult to appreciate when explained by words and static diagrams.

An excellent appreciation of a system can be gained by "hands on" experimentation.² This principle was found very helpful by the investigator while learning LISP: a system [9] was available which would step through programs, back up and restart, change values, and so on. This type of environment forces the student to think and act about what is happening, and can quickly lead her to a basic and confident appreciation of a system. By making the animation system interactive, the student

---

¹It is interesting to note that the operation of hardware is often represented graphically by means of an oscilloscope.

²This is the motivation behind LOGO, a simple programming language which teaches mathematical and programming skills by being "played with" by children [8].

could explore the LISP world visually and at her own pace.

The programming language LISP is quite different from most programming languages. S-expressions and EVAL are concepts which are unique to LISP, and a relatively experienced programmer may feel that she is a beginner when she tries to learn LISP. Because it depends on these abstract concepts, LISP was selected as the language to be animated.

1.3 RELEVANT LITERATURE

Although computer animation has been fairly popular for a number of years, few attempts have been made to produce animated computer science films. Until recently, only one significant computer animated film has been produced which is related to computer science [15]. (This film was produced by a system which cost approximately \$600 per minute of film [14].)

Baecker and his students have done work in this area which Baecker has aptly named "program illustration" [2]. Two systems have been produced by them. One system animates any program written in a subset of LOGO [2, p. 160]. A fixed set of conventions determines how the execution of statements and evaluation of variables will be displayed visually. Optional parameter settings may be added to the program to tailor the timing and spatial positioning of the animation. The system has been used to animate a simple LOGO program which reverses a character string, demonstrating features of the LOGO language as well as the concept of recursion. This system allows any LOGO program to be animated without adding special commands. The system is too expensive for production work, however.

The second system animates programs written in a subset of PL/1 [2, p. 161]. Psuedo-comments, interspersed with the PL/1 source code, call special functions which produce the animation. Producing a film with this system may take several hours of programmer effort since the pseudo-comments must be written for each program to be animated. However, more detailed control of the animation is possible than in the LOGO system. This system has been used to produce a film illustrating a sorting algorithm; the film followed the execution of the PL/1 program without making reference to the PL/1 language. The PL/1 system can produce film clips for \$100 to \$200 per minute.

Both of these systems produce key frames which are used by a computer animation language to produce the final film. (See Section 1.5.1.) Neither system can be used interactively.

Hopgood has produced computer animated films illustrating hash table algorithms [13]. His system applies various algorithms to examples which are too large to be managed easily by hand, thus giving students an appreciation for the methods when applied to non-trivial problems. This is claimed to show the advantage of certain algorithms more clearly than mathematical analysis or simple examples.

1.4 TYPES OF ANIMATION

Four major areas in computer science are likely to benefit from animation. The methods developed are likely to differ, and when "program animation" is mentioned, the exact type should also be specified. The areas are:

1. Animating algorithms, such as sorting, parsing, or searching.
2. Animating programming language features, such as "DO" in FORTRAN or "COND" in LISP.
3. Animating hardware operation. Animating movement of data between registers, arithmetic units, memory and channels.
4. Animating concepts in computer science, such as control structures and data structures.

1.5 ANIMATION METHODS

1.5.1 Key Frame Animation

Hand animation and computer animation both have made use of the key frame animation technique. An image is first produced for every "key" frame of the finished movie, and then "less skilled" artists (or a computer) produce intermediate "interpolated" frames, resulting in the smooth movement of the figures from one key frame to the next. In the case of computer key frame animation systems, the key frames could be produced by an artist using a data tablet, or by another program. The computer uses an algorithm which matches line segments in two successive key frames and then produces intermediate frames. The line segments of the first key frame are moved and lengthened or shortened gradually until they are transformed into the matching line segments in the second frame.

Key frame animation is very suitable for artistic films, where there is no need for precise control of the interpolation between key frames. The key frames may have different numbers of lines and are usually not drawn to exact specifications, and so a generalized rule is necessary to produce adequate animation in every case. A generalized rule usually will make assumptions and approximations, however, and may take more computation time than a special-purpose rule. Key frame animation may be useful in producing computer science films, as Baecker has shown [2], but it is not the only possible method.

1.5.2 Simple Animation

Simple animation relies on the facilities of a graphics system to produce animation. The tools developed for interactive graphics are extended to produce a sequence of animated displays. Many graphics systems provide means to displace, scale, rotate and change the intensity of sets of vectors. These facilities can be used to produce movement and other animation effects. The details of producing animation in this way are more involved than in key frame animation: timing considerations and display file organization may be difficult, and complex motions of artistic figures may be impossible. On the other hand, since most of the detailed animating is done at high speeds by the display processor, it is generally more feasible to produce real-time "live" animation by this method. A frame can be set up by the main computer and movement initiated. While the display processor is producing the

movement the main computer can be working on the next major display change.

CHAPTER II - SYSTEM DESIGN

2.1 DISPLAY DESIGN

The overall purpose of the ANTICS system is to impart some knowledge of the basic operation of LISP to a beginning programmer. There are two fundamental concepts which form the foundation of LISP: the structure of S-expressions, and the operation of EVAL. The design of ANTICS is centered on these two concepts.

2.1.1 S-expressions

Although S-expressions occasionally are changed during the execution of a typical LISP program, they generally specify "structure" in some sense, and the first thing a student must do is understand S-expressions as static data structures. It is important to understand how S-expressions map into CONS-cells in order to have a thorough knowledge of LISP, especially in the case of circular lists. ANTICS provides the capability of displaying arbitrary S-expressions either in "prettyprinted"³ form or as CONS-cells and pointers. Figure 1 shows a CONS-cell display. (The script which produced this display is included in Appendix B.)

---

³Prettyprinting is a style of printing LISP functions which makes them easy to read. All of the LISP examples in this thesis are in prettyprinted form.

1 - (SETQ Z '((A) B))

3 - (RPLACA (CDR Z) (CAR Z))

2 - (RPLACD (CDR Z) Z)

4 - (RPLACD Z Z)

Figure 1 - An Example of the Display of S-expressions
(actual size)

Since S-expressions are basically static, dynamic animation is not particularly applicable to them. A solid understanding of S-expressions may be obtained by examining a number of examples, and this is the approach taken in the design of ANTICS.

2.1.2 Evaluation

The operation of EVAL is a complex, dynamic process. It consists of a precisely ordered sequence of recursive operations on S-expressions, coordinated with the binding and unbinding of variables on a stack or a-list. It is possible to be an adequate LISP programmer without knowing explicitly about EVAL, and the beginning student does not need to know about its existence. On the other hand, the student who is learning LISP as "just another programming language" is really being shown the various aspects of EVAL, and should appreciate the idea of EVAL when it is presented.

ANTICS bases its animation of LISP functions on the operation of EVAL, though it does not presume that the student has any knowledge of EVAL itself. ANTICS' animation simply reflects the various facets of EVAL as they are brought into operation, by displaying them in a suitable fashion. The details of the design have centered on selecting the parts of EVAL to display and devising suitable graphic representations for them. Not every step in the EVAL process has been animated; the fetching of the LAMBDA-expression has been omitted, for example. Other facets of EVAL may be included in the animation at the discretion of the user, such as the display of variables,

the stack, and "EVLIS"⁴ values.

The basic animation proceeds as follows (refer to Figure 2):

1. The form which is being evaluated is printed on the display screen and, after a short interval, a box is displayed containing the word "EVAL", indicating that the form has been sent to EVAL.
2. If the form contains parameters which must be evaluated, then their evaluation is animated to the right of the present display and the return values are moved into a column below the present "EVAL" box. After these "EVLIS" values have all been returned, their bindings to the dummy variables of the LAMBDA-expressions optionally are displayed and the old values of the dummy variables are placed on the stack. Whenever the center of activity moves from the EVAL display to some other part of the screen, such as the stack display, the viewer's attention is directed by moving a "follower" to the new area of the screen. The follower is simply a star or other simple figure which moves to the peripheral display area at a rate which is followed easily by the viewer's vision.
3. If the function being evaluated is part of the LISP system⁵ the returned value is displayed after removing the original form from the screen. If a user-defined

---

⁴EVLIS refers to the function that evaluates arguments to LISP functions. It is part of EVAL [16].

⁵A SUBR, a function written in assembly language.

MEMBER:

```
(LAMBDA (THING LIST)
  (COND ((NULL LIST) NIL)
        ((EQUAL THING (CAR LIST)) LIST)
        (T (MEMBER THING (CDR LIST)))))
```

```
(MEMBER 'A '(C A T))
  THING = A
  LIST = (A T)
```

```
(COND ((NULL LIST) NIL)
      ((EQUAL THING &) LIST)
      (T (MEMBER THING &)))
```

```
(MEMBER THING (CDR LIST))
```

```
(COND ((NULL LIST) NIL)
      ((EQUAL THING &) LIST)
      (T (MEMBER THING &)))
```

```
(EQUAL THING (CAR LIST))
  EVAL → A
  EVAL → (CAR LIST)
  EVAL → LIST
  EVAL →
```

```
LIST = (C A T)
THING = A
LIST = *UNDEF*
THING = *UNDEF*
```

Figure 2 - Snapshot of EVAL Animation

(8/10 actual size)

function is being evaluated, the expressions which constitute the body of its definition are evaluated in turn. Expressions which are conditions in a COND or SELECT function are animated to the right of the current EVAL box, while all other expressions are animated so as to appear directly below it.

4. Throughout the animation, the definition of a function may be displayed at the top of the screen, and the portion of it that is currently being animated will be intensified. The viewer can tell easily at any time exactly how the form currently being evaluated relates to the overall function definition.
5. The LISP stack or association list⁶ optionally may be displayed on the screen, as well as current variable bindings. Changes in these are shown as they occur.
6. Whenever the EVAL animation or the stack display encounters the border of the display area, part of the display is "rolled off" the screen and disappears, thus freeing space on the display screen. When the evaluation is exiting the rolled portions reappear on the screen.

2.2 SINGLE-FRAME VS. REAL-TIME ANIMATION

A computer system to produce general-purpose, high quality animation probably would have to produce one frame at a time,

---

⁶LISP systems commonly use either a stack or association list (commonly called the "a-list" [16]) to save variable bindings. ANTICS will animate either method.

given the present state of the development of computer graphics systems.

A high priority in the design of ANTICS was that the system be able to produce animation in "real-time." This means that someone could look at the display screen and see the animation exactly as it would appear in a finished film. There were several reasons for this:

1. The implementation of ANTICS would be very difficult if all testing had to be done by filming one frame at a time.
2. The production of properly timed films for classroom use also would be very difficult.
3. A real-time system also could be made interactive, and this would provide an extremely versatile, though perhaps expensive, teaching tool.
4. Many animation systems have produced one frame at a time. It would be valuable to help to demonstrate that this is not always necessary.
5. A real-time system easily can be made to display one frame at a time if this is needed to produce high quality film.

The goal of producing a real-time system caused some implementation difficulties, but these were considered minor in light of the first two reasons given above. In addition, single-frame animation is much more complicated or impossible if the scan time of the display screen is longer than the longest camera shutter time, and this is the case with ANTICS and the available equipment. (See Appendix A, Section 8.)

2.3 SCRIPT DESIGN

It was assumed that a person making a film with ANTICS would be a relatively sophisticated LISP user (a course instructor or assistant), and that the instructions necessary to produce a film (called the script) would not need to be especially simplified. On the other hand, since the system would also be interactive, a relatively naive LISP programmer should be able to produce instructive animation with only a line or two of input. The result was the following design:

1. A single command, #EVALQ, will animate the evaluation of a function once it has been defined.
2. Several simple commands control the other basic displays. (#SHOW, #STAR, #DISPLAY and #DEFUN.)
3. Additional commands provide more elaborate control over details of the animation.
4. A sophisticated user may insert "breakpoints" into the definitions of the functions to be evaluated. The breakpoints can conditionally change minor details in the animation.
5. A general-purpose graphics language is available to build special displays such as titles, explanatory text and figures. This language is compatible with a subset of GLISP [11].

The choice of the syntax of commands is limited by the syntax of LISP. Care was taken to make the order of parameters and selection of default values as consistent and logical as possible.

Timing information could be indicated in the script in two

ways: by including an absolute starting time as part of every command, or by providing a command to wait a given time interval. The second method was chosen because it was simple to implement and because it made the reorganization of scripts much less complicated.

The types of commands that are allowed in the script can be broken down into several groups: one group controls the major displays; a second group controls details of the animation; a third group is concerned with details of producing films, such as timing and camera control.

The reader should refer to Appendix A, Section 1 (page 39) which contains a simple annotated script.

2.4 GRAPHICS PRIMITIVES

In order to provide flexibility to the user of ANTICS, a general purpose graphics language was provided. The following points were considered:

1. The user would most likely want to produce titles, diagrams, and explanatory text. She also may want to highlight portions of the animation by adding special effects such as pointers or outlining boxes.
2. It should be possible to include figures sketched with the light pen or data tablet.
3. The graphics language should be easy to learn.

An obvious choice was to base the graphics language on a subset of GLISP, a LISP based graphics language [11]. GLISP has a useful set of primitive functions for drawing lines and text.

It also allows the user to sketch figures with the light pen or data tablet and to adjust their size and position on the display screen, and also to save figures in a library. Furthermore, GLISP is supported and well documented.

The GLISP primitives and data structures are in the form of S-expressions, and their adaptation to ANTICS was simple. It also was possible to use the GLISP primitives while writing parts of ANTICS. Additional primitives were added to produce a fade effect and "invisible hand drawing." (Appendix A, Section 6 (page 45) contains detailed instructions on the use of the graphics language. Appendix A, Section 10 (page 60) is a reference list of graphics primitives.)

2.5 HUMAN FACTORS AFFECTING DESIGN

Since ANTICS is an educational tool, effort was made to consider its design in this respect. At one level, the graphic representation of concepts falls under this category. At another level, the timing and sequencing of the animation were considered. ANTICS also is an interactive graphics system, and this was taken into account in its design.

The graphical representation of S-expressions is a well established standard form, and there was not much latitude in its design. The method of animating EVAL is not standard however; it is based on a "blackboard" method used in a university course on LISP programming.⁷ This representation was

---

⁷CPSC 509, taught by Raymond Reiter at the University of British Columbia, Vancouver, Canada.

found to be instructive by the investigator and by others taking the course. The well-defined nature of EVAL tends to limit the possibility of radically different graphic representations. The real choices lie in the complexity of the functions animated and the amount of detail shown, and ANTICS provides the instructor/ animator the ability to match these choices to the level of her students.

The timing and sequencing of animation were important considerations in the design of ANTICS. Early versions of the program produced animation that was difficult to follow because the "action" moved from one side of the display screen to the other without warning. The idea of a "follower" was developed: a moving figure on the screen naturally catches the eye and directs the viewer's attention to a new area of the screen. Movement was found to be very compelling visually -- in fact, any movement on the screen seems to lock the viewer's attention. Because of this, the rule for lengthening ANTICS' animation gives priority to making the static portions of the film longer, since these sections allow the user to absorb the meaning of the animation sequence. Movement is a dominating element in entertainment animation, but its use in instructional animation must be considered carefully [10, p. 66].

The overall speed of the animation must be slow enough for the viewers to follow, and this speed is not easy to define. No exact speeds can be recommended, but the following points were considered:

1. Viewers gain skill at following the animation as they

become more familiar with the form of representation used in the animation.

2. For naive LISP programmers, "the slower the better" is perhaps a very realistic rule for the speed of animation. (On the other hand, an authoritative source claims, "It has been proved by experience that the shorter the film the more effective the instruction is likely to be, because of the intense concentration which the student must give. . . ." [10, p. 136])
3. A short film can be shown several times consecutively and different facets of LISP can be emphasized each time by the instructor.
4. A suitable projector can stop the animation while the instructor makes detailed explanations.⁸

The speed of animation can be adjusted with the #RATE command. A one-minute film can be stretched to approximately ten minutes by adding the command (#RATE 0.1) to the ANTICS script. One way for an instructor to decide how fast animation should be is to ask herself how much lecture time she would devote to the subject and then to scale the film accordingly. (Alternatively, she might base the animation speed on cost. Section 3.4 has details of the cost of producing films.)

Two important considerations in the design of interactive

---

⁸If a single frame of film is intended to be projected, non-reversed (negative) film should be used, giving black lines on a white background. Film with a black background is much more likely to melt when the projector is stopped.

graphics systems are response time and methods of interaction. ANTICS' response time has been excellent due to the design of the supporting system, and no design decisions were affected by it. (See [17] for an analysis of system response times.)

Newman and Sproull recommend that interaction be primarily through a single input device as this is more graceful for the user [18]. ANTICS primarily uses the Adage function buttons, since they most nearly match the characteristics of the majority of the interactive requests: simple actions without parameters. (A light pen menu also fits this application, but was not considered seriously since the display screen must be left clear for filming.) In special cases other input devices were used. A dial was used to adjust the animation rate since it is easy to learn the relationship between the dial position and the rate. The light pen was used for "backing-up" the animation and selecting variables to be changed since pointing is a natural way to accomplish these operations. The keyboard was used to enter animation commands and new atomic values since it was the only practical alternative. (Appendix A, Section 3 (page 42) describes how to use the interactive features of ANTICS.) Interacting with several devices in succession can be confusing for the user, but in ANTICS, which is primarily an animation system, the methods described were found to be natural to use, and the goal of providing a single interactive medium was not given high priority.

CHAPTER III - IMPLEMENTATION

This chapter is concerned with important aspects of the implementation of ANTICS. It does not cover every detail -- it refers mainly to the factors which affected the overall structure of the implementation and which may be of interest to future implementors of similar systems.

3.1 SYSTEM ORGANIZATION

The implementation of ANTICS is dependent on a unique environment of hardware and software. This is unavoidable due to the interactive nature of ANTICS: interactive graphics systems tend to be hardware dependent. The general organization of the implementation environment is fairly typical of graphics systems however, and it may be possible to modify ANTICS to work on other systems without undue effort.

3.1.1 The Adage Graphics Terminal

The ANTICS system is implemented on a Model 10 Adage Graphics Terminal which communicates with an IBM 370/168 computer operating under the Michigan Terminal System [3]. The user communicates with the 370 through an IBM 3270 Display Station which is located next to the Adage display screen. Refer to [4] for more information.

3.1.2 The GRAPH Supervisor Program

The Adage Graphics Terminal contains a supervisory program named GRAPH which simultaneously controls the refreshing of the display screen and handles communication with the IBM 370/168 [6]. The supervisor contains a 6000 word display buffer. Each word in the buffer either causes a vector to be drawn on the display screen or causes one of thirty-eight control functions to be performed, such as scaling the size of the following vectors, displacing the X and Y axes of the display, branching to a new location in the buffer, enabling or disabling the light pen, modifying other control words, etc. The supervisor places commands into this buffer at any time under control of the IBM 370/168, and returns to the 370 any user input from the dials, light pen, or function buttons which are attached to the Adage Graphics Terminal.

3.1.3 LISP BASIC

LISP BASIC is a package of LISP-callable routines which communicate with GRAPH through the standard FORTRAN interface, AGTBASIC[7]. The LISP BASIC routines [12] assemble Adage vectors and control words, translate text into display vectors, send blocks of vectors and control words to the Adage Graphics Terminal to be displayed, and read the Adage buttons, dials and light pen. LISP BASIC provides only the most primitive control of the Adage Graphics Terminal.

3.1.4 LISP/MTS

ANTICS is written in LISP/MTS [20], an interpreter similar to LISP 1.5 [16]. LISP/MTS uses an internal stack rather than an a-list as do LISP 1.5 and several other LISP systems.⁹ The animation which is produced reflects this aspect of LISP/MTS and several other minor details, but since ANTICS contains its own EVAL function these details could be changed easily. The system probably would be significantly cheaper to run if a LISP compiler were used.

3.2 DISPLAY FILE ORGANIZATION

The organization of the contents of the Adage Graphics Terminal buffer, referred to hereafter as the display file, was altered several times in the course of the development of ANTICS. Newman and Sproull provide excellent information on the logical design of ideal display files [18]. In ANTICS the actual display file stayed fairly far from the ideal due to hardware and software limitations and design considerations. The chief factor was the design goal of keeping ANTICS real-time and interactive.

There are three basic limitations in the Adage graphics system which affected the display file organization in ANTICS. The first is that the Adage Graphics Terminal is relatively slow, since all characters are generated by software (in the IBM 370/168) and displayed as a series of vectors, and also since

---

⁹INTERLISP uses a binding method similar to an a-list [19, p. 12.2].

all display file commands (except vector commands) are interpreted by the GRAPH supervisor. An experimental version of the routine to display CONS-cell structures showed extreme flicker when the CONS-cell boxes were set up as displaced instances of one set of vectors which drew the box. When each box was displayed "in-line" by a separate set of draw commands the flicker was eliminated entirely. The flicker therefore was due to the overhead time of interpreting the displacement and jump commands in the display file. Because of this, the display file was designed to have a limited number of interpreted commands.

The second hardware limitation of the Adage Graphics Terminal is related to the way vectors are displayed: the X and Y coordinates of a vector command are absolute -- they may not be made relative to the previous vector. There are two registers, known as DX and DY, which are used to displace each vector, and these provide for movement or displacement of any set of vectors. There is no way to increment or decrement these registers, and therefore they can not be used to produce several levels of displacement. Therefore, any hierarchy built into the display file can have only one level of coordinate displacement provided by the system.

The third limitation in the Adage graphics system is that it is impossible to tell which instance of a subroutine call was selected by the light pen. For example, if all EVAL boxes were coded as one set of vectors which was displaced to various locations on the display screen, a light pen hit of a box would only indicate that a box was hit and not which instance of the

box. This problem can be solved only by a major revision of the GRAPH supervisor which would return a stack containing all jump addresses which had occurred before the light pen hit.

In addition to these hardware constraints, several design factors affected the display file organization. In order for the system to be interactive and real-time, the display may not be interrupted in order to reorganize the display file (to "garbage collect" it, for example). As a result, fixed portions of the display were set aside for sections of the ANTICS display file which varied in length, such as the atomic values displayed by the #DISPLAY command and the value returned from EVAL.

Another design constraint resulted from the need to be able to move parts of the display in order to provide animation. The design of the display file must take into account the relationship of vectors that will move together, as well as vectors which all must disappear simultaneously. In single-frame animation the display file can be reorganized when changes are made in the display to produce movement, but in a real-time or interactive system, the possibility of movement must be carefully taken into account at all. Also, since real-time movement is produced by slowly changing the displacement applied to a number of vectors, and the Adage graphics system only provides one level of displacement commands, the overall design of the display file must take into account the various uses of displacement commands.

Several false starts were made on display file design: Figure 3 illustrates the final design for the display file during the animation of evaluation.

ADDRESS:

1

SCALE	1.0
CAMERA	ON/OFF
NULL/JUMP	
LIGHTPEN	ON
INTENSITY	*
"STEP"	
INTENSITY	NORMAL
"AUTO"	
LIGHTPEN	OFF
JUMP	
SKIPZ	*
INCR	*
DY	*
LIGHTPEN	OFF
JUMP	(to stack)
LIGHTPEN	ON
INTENSITY	*
"X = "	
value	
JUMP	*
unused	
INTENSITY	*
"Y = " etc.	
INTENSITY	NORMAL
LIGHTPEN	OFF
box around variables	

Control
section

EVAL-RETURN

FWA-DISPLAY

#DISPLAY
command
section

Figure 3 - Display File Organization

ADDRESS:

27

FWA-STAR

#STAR
command
section

FWA-FOLLOWER

Follower
section

FWA-RETVAL

Return
value
section

Figure 3 - continued

ADDRESS:

EVAL-ROLL

JUMP-ROLL

(LOCLIST)

(STACKLIST)

6001

SKIPZ	*
INCR	*
DY	*
INTENSITY NORMAL	
DX	0
LIGHTPEN	ON
NULL/JUMP *	
EVAL display	
-----	
display produced by breakpoints	
-----	
temporary display such as EVLIS values	
JUMP	EVAL-RETURN
free space	
stack entry	
NULL/EOL	
stack entry	
NULL/EOL	
etc.	
EOL	

EVAL
animation
section

Stack
display
section

Figure 3 - continued

The control section of the display file always is present. It includes the basic scaling command, commands to control the operation of the camera automatically, and two light pen "buttons"¹⁰. It also contains a short section which provides a fixed buffer address for the displacement commands which roll the stack display, and a fixed address where the EVAL display can return.

The #DISPLAY command section is omitted if there is no display of variables. A fixed number of buffer words is allocated for each atomic value so that the display need not be reorganized when values change. Intensity commands with normal intensity values are inserted between the entries. These commands are changed when a variable is brightened to indicate that its value has been changed or referenced.

The #STAR command section also is optional. It contains the vector commands which display the prettyprinted "starred" function definition. Interspersed with these are intensity commands set to normal intensity. During animation of evaluation, these are replaced with bright intensity commands as the various parts of the function are evaluated.

The follower section contains the vector commands which draw the follower symbol and the necessary commands to turn it on or off and to produce movement. It is always present during evaluation animation.

The return value section always is present during animation

---

¹⁰A "light button" is an area of the display which may be selected by the light pen to trigger some action by the system. Usually it is a word describing the action.

and contains a fixed length area which holds the vectors which display return values. It also contains commands which are modified in order to move the return value and to make it disappear.

The EVAL animation section contains all the display commands producing the evaluation animation. Each level of evaluation adds to this section, and removes from it upon exiting. Displays produced by user breakpoints are placed here as well. This section begins with commands which are necessary to roll the display and to skip part of it when more space is needed on the screen. The section must end with a jump back to the control section so that the stack display is entered properly. During animation, LOCLIST is a list of addresses of the beginning of the Adage buffer locations where each level of the the EVAL animation has placed its display commands.

The stack display section starts at the end of the Adage buffer and exists only during evaluation animation. New stack display entries are added to the beginning of this area of the buffer. A jump to the beginning of this section is located in the control section. When parts of the stack display must disappear because it has been rolled partially off the screen, an end-of-list indicator is placed in one of the words reserved for this purpose. STACKLIST is a list of addresses in the display buffer where each stack entry starts. Stack entries are made by both LAMBDA-expressions and PROG function calls. When an a-list is displayed in place of a stack, a fixed number of buffer words is set aside for each entry so that the display can be changed if values are changed by SET or SETQ.

3.3 METHODS USED IN THE PROGRAM

The heart of ANTICS is a set of LISP functions constituting a version of the LISP EVAL function. (See [16] for a definition of EVAL.) This version of EVAL is interspersed with calls to animation routines, and this is how all animation is produced. The stack, variable and "star" displays are also driven by ANTICS EVAL, and user breakpoints are detected and processed by it. As a result, the animation naturally follows the execution of EVAL and ANTICS has a simple underlying structure.

The highlighting of the "starred" function by intensifying parts of it as animation progresses is accomplished by inserting breakpoints into its definition which are similar to user breakpoints. These contain the Adage buffer address where a word has been reserved; ANTICS EVAL intercepts these breakpoints and places an intensify command in the buffer location. (Both types of breakpoints must be detected and ignored at many other places in ANTICS.)

Interactively "backing up" the system is done by using the LISP/MTS UNEVAL function to return to higher levels of ANTICS EVAL, and clearing part of the display.

The routine which displays CONS-cells also uses some novel methods. When given a list it draws a CONS box, and then calls itself recursively with the CAR of the list. The return value is a pair of dimensions indicating the physical size of the display which was generated. These dimensions are used to locate the display of the CDR which is generated by a second recursive call.

3.4 COST AND EXECUTION TIME

It was originally assumed that ANTICS would be expensive to use since it combined two relatively expensive items in terms of computer charges: LISP and graphics. Development and debugging costs were not particularly low, but the cost of producing animated films has turned out to be surprisingly low. A three minute-twenty second film illustrating the execution of the recursive function MEMBER was produced for a total computation cost of \$4.23 utilizing 5.4 seconds of central processor time.¹¹ (The script for this film is included in Appendix B.) The cost of film and processing was \$7.50. The example should probably have been slowed down two or three times to make it easier to follow. This would only add a slight additional computation charge, as only the elapsed time and virtual memory usage would increase.

---

¹¹The cost could be reduced even further if a compiled version of ANTICS was used rather than the present interpretive system.

CHAPTER IV - CONCLUSIONS

4.1 COMPARISON WITH EXISTING SYSTEMS

Few program animation systems have been developed, and it is difficult to compare ANTICS to those that exist. ANTICS' strong points -- its cheapness and interactive capability -- are not found in any other systems. Also, existing systems have been designed to present subject areas quite different from ANTICS'. One thing that can be said is that ANTICS is not a general system in the same sense as the PL/1 animation system described in [2]. ANTICS' primary use is to teach features of LISP. After LISP is mastered and students are familiar with the style of ANTICS' presentation, the system could be used, perhaps, to illustrate general properties of algorithms. (For example, by animating EVAL itself. A script for this is included in Appendix B.)

4.2 POSSIBLE IMPROVEMENTS TO ANTICS

An improvement which could be added to the present system would permit more complex functions to be animated. The 6000 words of the Adage Graphics Terminal buffer are filled fairly quickly. (The display of Figure 2 (page 12) uses approximately 2900 words.) In the current implementation of ANTICS, when part of the display has rolled off the screen it remains in the buffer. This part of the display buffer could be saved in the

IBM 370/168 memory and the Adage memory could be made available. It would not be easy to accomplish this without blinking the display, since the contents of the Adage buffer would have to be reorganized. Since the usefulness of long and complex animations seems somewhat doubtful, this extension was not considered in detail. If this modification were made, another modification probably should be made to the routines which roll the display, allowing them to roll to the left as well as upward.

4.3 SUMMARY

A system has been implemented which animates LISP programs. The system can be used to produce animated films, film strips and slides, or it may be used interactively as an instructional aid. The cost of using the system is very low compared to other systems. The system will operate only on a specific hardware configuration, but elements of its design could be useful to other possible implementations.

BIBLIOGRAPHY

- [1] Baecker, R. M., Towards Animating Computer Programs: A First Progress Report, Proceedings of the Third National Research Council Man-Computer Communications Seminar, May, 1973, 4.1-4.10.
- [2] Baecker, R. M., Two Systems Which Produce Animated Representations of the Execution of Computer Programs, SIGCSE Bulletin, Feb., 1975, Vol. 7 No. 1, pp. 158-167.
- [3] Choit, M. D. (ed.), The MTS Commands Manual, University of British Columbia Computing Centre, 1970.
- [4] Coulthard, W. J., Operator's Manual for the Adage Graphics Terminal, University of British Columbia Computing Centre, 1970.
- [5] Coulthard, W. J., UBC CAMERA - Use of a Camera at the Adage Graphics Terminal, University of British Columbia Computing Centre, 1972.
- [6] Coulthard, W. J., UBC GRAPH - A Simple Interactive Graphics Package, University of British Columbia Computing Centre, 1971.
- [7] Coulthard, W. J.; DeKleer, J., UBC AGTBASIC - Basic Communications Package for the Adage Graphics Terminal, University of British Columbia Computing Centre, 1971.
- [8] Feurzeig, W.; Papert, S.; et. al., Programming Languages as a Conceptual Framework for Teaching Mathematics, Bolt Beranek and Newman Inc. Report 1889, 1969.
- [9] Friedman, P., An Interactive Debugging Package for LISP/MTS, Master's Thesis, University of British Columbia, Department of Computer Science, 1973.
- [10] Halas, J.; Manvell, R., The Technique of Film Animation, Communication Arts Books, New York, 1971.
- [11] Hall, W.; Jervis, B.; Jervis, J., GLISP - A LISP Based Graphic Language, University of British Columbia Department of Computer Science.
- [12] Hall, W.; Jervis, B.; Jervis, J., LISP BASIC - A LISP-AGTBASIC Interface, University of British Columbia Department of Computer Science, 1973.
- [13] Hopgood, F. R. A., Computer Animation Used as a Tool in Teaching Computer Science, Proceedings of the 1974 International Federation of Information Processing Societies Conference, North-Holland Publishing Co., 1974, pp. 889-892.

- [14] Knowlton, K. C., Computer Produced Movies, Science, Nov. 26, 1965, Vol. 150, p. 1116.
- [15] Knowlton, K. C., L6: Bell Telephone Laboratories Low Level Linked List Language, two black and white sound films, Bell Telephone Laboratories, Murray Hill, N.J., 1966.
- [16] McCarthy, J.; Levin, M. I., et. al., LISP 1.5 Programmer's Manual, M.I.T. Press, 1962.
- [17] Miller, R. B., Response Time in Man-Computer Conversational Transactions, Proceedings of the Fall Joint Computer Conference, 1968, Vol. 33, pp. 267-277.
- [18] Newman, W. M.; Sproull, R. F., Principles of Interactive Computer Graphics, McGraw-Hill, New York, 1973.
- [19] Teitelman, W., et. al., INTERLISP Reference Manual, Xerox, Palo Alto Research Center, 1974.
- [20] Wilcox, B.; Hafner, C., LISP/MTS User's Guide, University of British Columbia Department of Computer Science, Sept., 1974.

APPENDIX

APPENDIX A - ANTICS USER'S GUIDE

USING THIS GUIDE

Reading Sections 1 and 2 is a bare minimum for using ANTICS. A casual user will want to read Sections 1 through 5, and an advanced user or film-maker should read all sections. All users will want to refer to Section 9, ANTICS commands. This guide presupposes some familiarity with LISP/MTS [20], MTS [3] and the Adage Graphics Terminal [4].

1. THE ANTICS SCRIPT

The input to ANTICS is a series of LISP forms called a script. When a movie is being produced this script must be entirely on an MTS file so that it can be read without interruptions, which would confuse the timing of the movie. There is no reason why the script cannot be entered one line at a time when the system is being used interactively or while experimenting with ideas. A script may consist of several lines to produce an animation of a single LISP evaluation, or it may contain a hundred or more lines to produce a complete movie with titles and explanatory text. The following script produces a rudimentary animation:

```

(#START)

(DEFUN MEMBER (THING LIST)
  (COND ((NULL LIST NIL)
    ((EQUAL THING (CAR LIST)) LIST)
    (T (MEMBER THING (CDR LIST))))))

(#DISPLAY THING LIST)

(#WAIT 5)

(#EVALQ (MEMBER 'A '(C A T)))

```

#START initializes some parameters and blanks the display screen. DEFUN simply defines the function of interest in the usual way. #DISPLAY causes the names and values of the atoms THING and LIST to be displayed on the screen. They will remain until the screen is blanked, and the display will be changed whenever their values change. #WAIT causes a five second pause before the next line of the script is executed. #EVALQ causes a complete animation sequence, lasting perhaps several minutes, of the EVALuation of the given form. During this animation the values of the atoms specified by the #DISPLAY command are updated constantly.

There are two additional major commands which could be added to the above script anywhere before the #EVALQ command:

```

(#STACK THING LIST)

(#STAR MEMBER)

```

#STACK causes a stack to be displayed during animation showing the bindings of the atoms THING and LIST. No display is produced when the #STACK command is given -- the display is shown during the animation produced by #EVALQ. Optionally, an a-list may be displayed in place of the stack. #STAR causes the

definition of MEMBER to be shown immediately at the top of the display screen. During the animation produced by #EVALQ the portion of this "star" function currently being evaluated will be intensified.

These commands are the basic high-level animation features of ANTICS. The user can obtain a great deal of variety, however, by selecting options, changing parameters, and producing additional graphical displays with the graphics primitives included with ANTICS.

2. HOW TO RUN ANTICS

Before attempting to use ANTICS, make sure that the GRAPH supervisor program is running in the Adage Graphics Terminal. GRAPH usually is ready; if it is necessary to restart it, see [6]. (Either of the current versions of GRAPH will work.) ANTICS may be run by executing the MTS command:

```
$SOURCE MASD:ANTICS
```

while signed on at the 3270 Display Station at the Adage Graphics Terminal. A #START command is executed automatically and the user may define a function and use #EVALQ immediately to see the animation. The PF9 key is set to "(#BLANK)" and may be used to clear the screen. The PF6 key is set to "NIL" and may be used to exit from BREAK loops. A script may be read by using the MTS command:

```
$CONTINUE WITH script RETURN
```

where "script" is the name of a file containing the ANTICS commands comprising the script. If a medium length or longer script is used, the 3270 Display Station may go into pause mode

when the return values of the commands are listed. This may be avoided by placing a PROGN call around all of the commands or by using the %FAST device command to prevent the 3270 from pausing.

3. USING THE SYSTEM INTERACTIVELY

Although ANTICS is designed primarily for making movies, it is fast enough to support interactive use. Interaction is by means of the light pen and function buttons. (An overlay card is provided which labels the function of the various buttons.) Whenever ANTICS is waiting, both of these devices are active. During animation, #WAIT is called before and after evaluating each form.

There are two modes of interactive operation: STEPMODE and AUTOMATIC. STEPMODE is selected by pressing the STEPMODE button (15) or by using the light pen in any way. While in STEPMODE, the function #WAIT always waits until either the STEP button (13) is pushed or the STEP light button is selected with the light pen. The STEP light button is intensified while ANTICS is waiting. In AUTO mode, the #WAIT function waits the specified time, unless the STEP button is pushed first. AUTO mode is selected by pressing the AUTO button (16) or by pointing the light pen at the AUTO light button.

The light pen has two other functions as well. At any time it may be pointed at any variable in the list created by #DISPLAY, and a new value for the atom may be entered through the keyboard. The light pen also may be pointed to any part of the evaluation display, and the animation will back up to that point and restart.

There are several other buttons which may be used at any time. The QUIT button (10) causes a return to the top level of LISP. This will terminate an animation sequence and proceed to the next command in the script. The animation display is not erased. The PLOT button (9) will plot whatever is on the display screen at the time it is pushed. It is identical to the command #PLOT. The BREAK button (11) enters a LISP/MTS break loop for debugging purposes. If this button has been used, the animation may be resumed by entering NIL from the keyboard or by using the PF6 key. The CAMERA button (12) turns the camera on or off alternatively, using relay 1. The BACKUP button (5) causes the last form visible on the display of animation to be re-evaluated and animated. This button is enabled only during evaluation animation.

The RATE button (14) reads a value from the Adage dial "A" and calls the #RATE command to change the rate of animation. This can be used to experiment with various speeds of animation; the #RATE value which was used is printed and a #RATE command later can be inserted into the script when an appropriate value has been established. High #RATE values can be used to preview an animation quickly. (Note that the light pen has trouble responding when the animation proceeds too quickly. When this happens, the STEPMODE button may be used before the light pen is used.)

4. THE ANTICS EVAL FUNCTION

In order to produce animation, ANTICS contains its own EVAL

function which is interspersed with calls to graphics routines. This EVAL contains some peculiarities and limitations of which the user of ANTICS must be aware. None of these are restricting to the kind of LISP functions which are likely to be animated. In most respects, ANTICS' EVAL appears like LISP/MTS rather than other LISP systems.

The basic restriction is that the ANTICS EVAL uses its own internal a-list to store atomic values. (See [16] for a description of the use of the a-list.) Therefore, ANTICS atoms and LISP/MTS atoms are completely independent (except when CAR and RPLACA are used on atoms -- they will refer to LISP/MTS atomic values). This also means that FSUBRS and NSUBRS all must be explicitly coded into the ANTICS EVAL, or else they may try to get atomic values from LISP/MTS in the normal fashion. The following functions, normally FSUBRS or NSUBRS have been coded into the ANTICS EVAL:

COND	AND	OR	SELECT
PROG	SET	SETQ	SELECTQ

The following commonly used FSUBRS and NSUBRS are not available during animation:

DEFUN	UNCONS	DEFINE	LABEL	DEFPROP
UNTIL	WHILE	DO	STATUS	

Additionally, two functions have been provided for use in the script: #SET and #SETQ. They appear to work exactly as SET and SETQ normally do in LISP/MTS, except that they work with the ANTICS EVAL a-list. If a variable is set which is not yet on the a-list, it is appended to the a-list. (This is different from LISP 1.5.)

If EVAL or APPLY are called within a function being animated, the ANTICS EVAL or APPLY is used.

RETURN and GO should only be used directly within a PROG. RETURN may not take a second parameter. GO must be given a label name, it will not call EVAL.

PEXPRs, NEXPRs and no-spread functions may be used, as well as the function ARG. (Calls to ARG may have only one argument.)

If a function is undefined then the value of the function name is taken, and if it is not NIL it is treated as the function definition (as done in LISP 1.5).

The various MAP functions should not be used.

Since ANTICS EVAL does not contain all of the normal LISP error messages, it is important to be sure that a function is correct before attempting to animate it.

5. ADDITIONAL COMMANDS

The command #DEFUN has exactly the same effects as LISP DEFUN, and in addition it causes a display. The DEFUN statement itself is displayed at the top of the screen, and the CONS-cell structure is displayed below.

The command (#SHOW S X Y) displays the CONS-cell structure of the given S-expression on the screen, starting at coordinates X,Y. Circular S-expressions may be displayed in this manner. These and other commands are explained in detail in Section 9.

6. GRAPHICS LANGUAGE

There is a general purpose graphics language available to

the user of ANTICS. It contains both graphics primitives and special functions for displaying LISP structures. This graphics language can be used directly in the script to produce a variety of displays within the movie, such as text and diagrams. It also can be used within special breakpoints which may be inserted into a function whose evaluation is being animated. By using these breakpoints, which are invisible to the viewer of the movie, the user of ANTICS may tailor the evaluation animation by omitting unnecessary detail and displaying explanatory messages and figures at key points in the animation. Details of the individual primitives are found in Section 10.

6.1 GRAPHICS LANGUAGE WITHIN THE SCRIPT

When a command is included in the script, the graphics display which results is shown automatically on the screen. When graphics primitives are used, however, they must be embedded in the command #DISP in order to be displayed. For example:

```
(#DISP (*SYMBOL -2.5 .6 'HELLO 1.)
```

```
(*SYMBOL -2.5 -.6 'THERE 1.))
```

may be included in the script to display the message on the screen. The message will remain on the screen until the next #BLANK command. Successive #DISP commands will cause a display to be built up on the screen, and the #WAIT command may be used to control the timing. The FOR or FADE primitives may be used if it is desired to have part of the display turn off before using the #BLANK command to clear the entire display. Even if FADE or FOR have caused the entire screen to be blanked, #BLANK

should be used so that all locations in the display buffer are made available.

6.2 GRAPHICS LANGUAGE EMBEDDED IN FUNCTIONS TO BE ANIMATED

Breakpoints may be inserted into any function to be evaluated. The form of a breakpoint is:

```
(# ENTRY-FORM FORM EXIT-FORM)
```

FORM is part of the function which is being animated. Before FORM is evaluated and animated, ENTRY-FORM is evaluated by LISP/MTS. ENTRY-FORM may contain graphics primitives and it also may change various parameters effecting the animation in progress. (See Section 11 for information on ANTICS variables.) After FORM is evaluated and animated, and the animation display is removed from the screen, EXIT-FORM, which is optional, is evaluated by LISP/MTS.

Graphics primitives may be included in ENTRY-FORM, and they will be displayed on the screen when the text of FORM is displayed. The display will remain on the screen until a return is made from the evaluation of FORM. If a shorter time is desired, the FOR primitive should be used. Graphics primitives used within EXIT-FORM should be included within the command

```
(#EXIT-DISP SECONDS primitives)
```

in order to be displayed for the given number of seconds. The following notes apply to breakpoints:

1. The variables X and Y are the coordinates where the text of FORM will be displayed as part of the animation. They may be used to locate displays generated by the breakpoint.

2. The variable `#TIMES` is bound to the total number of times the particular breakpoint has been entered within the current animation. It can be used, for example, in a conditional expression to make the breakpoint's action dependent on the number of times the viewer has been shown a particular detail.
3. Two variables, `VAR1` and `VAR2`, are protected by a `PROG` and may be used to pass values from `ENTRY-FORM` to `EXIT-FORM`.
4. The value returned from the evaluation of `FORM` is available to `EXIT-FORM` as `RETURN-VALUE`.
5. The animation may be turned on or off by setting `DISP?` to `T` or `NIL`. This may be done only in `ENTRY-FORM`. When animation is turned off, three dots are displayed on the screen as an indication. If `OLD-DISP?` is set to `NIL` in addition to setting `DISP? = NIL`, the three dots also will be omitted from the display.
6. An `ENTRY-FORM` can override the effect of a `#DEPTH` command by setting `DISP?` to `T`.
7. `FORM` may be omitted from a breakpoint. In this case, the breakpoint consists of `ENTRY-FORM` alone. This may be useful whenever a list of forms is being evaluated, as in `PROGs` or conditional statements. The value returned from `ENTRY-FORM` is returned, and will be significant within conditional expressions such as `AND` and `OR`.
8. A dummy LISP/MTS function named `"#"` is supplied which

simply EVALS FORM. This permits functions containing breakpoints to be evaluated by LISP/MTS for testing. If FORM is omitted from the breakpoint NIL is returned. (See Note 7 above.)

9. ANTICS EVAL atomic values may be obtained by calling the function *VALUE. The form is:

(*VALUE ATOM A ERR)

The value of ATOM is the name of the atom, A is the ANTICS a-list (named A), and ERR is evaluated and returned if ATOM is undefined.

7. INTERFACING WITH GLISP

Several of the graphics primitives in ANTICS are identical to low level GLISP functions (See [11]). This feature allows the user of ANTICS to create SUBROUTINES in GLISP and include them as part of her ANTICS script. One of the various GLISP sketching routines can be used to draw a picture which is then included as part of the ANTICS script, perhaps as part of the movie titles.

GLISP SUBROUTINES which use the following functions may be inserted directly into the ANTICS script:

*GOTO	*PLOT	*SYMBOL	*INTENSITY
*DASH	*UNDASH	*FOR	ADJ(without dials)
GPROG	*JUMP	*SKIPNZ	*SKIPZ

The SUBROUTINE name must be included, though it is ignored. SUBROUTINES may not be included within ANTICS breakpoints, although *GOTOS, *PLOTS and *SYMBOLS created by GLISP may be

included. Note that ANTICS does not treat SUBROUTINES the same way that GLISP does -- they are displayed when they are encountered in the script and they do not need to be called in any way. Also, when primitives created by GLISP are inserted into ANTICS breakpoints, it should be remembered that ANTICS' scale is set to 1.0 while GLISP runs with a scale setting of 0.5. The SUBROUTINE function in ANTICS automatically adjusts the scale to 0.5 when GLISP SUBROUTINES are included directly in the script.

The following GLISP SUBROUTINE produces a box which outlines the area of the screen where ANTICS produces animation, and where a Super-8 mm. camera should be pointed.

```
(SUBROUTINE FRAME
```

```
  (*GOTO -10. 10.)
```

```
  (*PLOT 10. 10.)
```

```
  (*PLOT 10. -4.8)
```

```
  (*PLOT -10. -4.8)
```

```
  (*PLOT -10. 10.))
```

8. PRODUCING FILMS

ANTICS can be used to make film strips, slides, or animated films. Film strips or slides can be made using STEPMODE (described in Section 3.) and simply photographing suitable steps in the animation sequence [5]¹². The following steps will

---

¹²The exposure settings given in this reference might be reduced to give better results.

produce a film with minimum flicker. This elaborate procedure is necessary due to the inherent incompatibility of the camera's shutter speed and the Adage refresh rate. The procedure is given as a guide for experimentsation. Some details may possibly be changed -- especially the Adage refresh rate and the exposure setting.

1. Press the RESET and then the RUN buttons on the Adage display console. Execute the MTS command:

\$COPY AGT:GRAPH >AGTI

2. Set the Adage refresh rate to 30 times/second:

- a. Set the switches on the Adage control console to 22000 01612 (octal) and press PULSE1. The lights in the button register should read 61014 00001 (octal). If they do not, AGT:GRAPH has been changed.

- b. Press RESET BR and set the button register to 61014 00002 by pressing the buttons (which also light up). (Setting the buttons to 61014 0000N will set the refresh rate to $120/(N+2)$ times/second.)

- c. Set the switches to 20200 01612 and press PULSE1. Set the switches to 20200 00754 and press PULSE1.

- d. Place the statement (SETQ FRAMES/SEC 30) at the beginning of your ANTICS script.

3. Set the f-stop on the NIZO S 56 camera to f/1.8. If the indicator needle fails to move the mercury batteries are dead. In that case, the shutter will stay open to f/1.8 and the film still will be properly exposed. Check the motor batteries by pressing the power switch forward -- if the exposure meter reads to the right of f/8 the batteries

are low. Set the camera speed to 18 frames/second. Set the filter for artificial light. Load the camera with Kodak "4-X Reversal Film 7277" (ASA 400 rating). The master intensity control knob located under the Adage display screen should be set with the mark pointing straight down.

4. Set the camera on a tripod about 3 to 4 feet from the Adage display screen.
5. Run ANTICS. (See Section 2.) Execute a #FOCUS command and adjust the camera by tilting the tripod and moving the zoom lens. Focus the camera carefully. Press the STEP button (13) to end the #FOCUS command.
6. If the camera is connected to the Adage relay 1 (on top of the display console), the #CAMERA command in the script will turn the camera on or off. See the example MEMBER in Appendix B for a good way to organize the script. If the camera is being controlled by the relay, leave the camera's power switch off until you have finished experimenting.
7. Run ANTICS and expose the film. The room lights should be turned off and the lights on the Adage control panel should be covered. A 50 foot cartridge lasts 200 seconds at 18 frames/second. To expose more than one cartridge, press the STEPMODE button (15), change film, and press the AUTO button (16). The camera will be stopped and restarted.

Alternatively, the film can be stopped automatically by initially setting the variable FILM-LENGTH to the number of seconds of film to be exposed. When ANTICS has turned the camera on for this length of time it will stop the

animation and the camera, print a message, and enter a LISP break loop. You may then change the film and type "NIL" to restart. If the new cartridge is a different length than the previous one you may change the value of FILM-LENGTH with SETQ before leaving the break loop. This feature will not stop the camera during the time that a #DRAW command is executing.

8. The actual elapsed time of a script seems to vary with system load. This can cause problems if you are trying to exactly fill a film cartridge. The #RATE command can be used to adjust the duration of animation.
9. If the ADAGE is given frames to display which take longer than 1/30 second, the frame rate will change and you may get an objectionable strobe effect or flicker on the film. The most complex frames in the MEMBER example, displaying 7 EVAL boxes, take almost exactly 1/30 second. (See Figure 2 (page 12). This figure represents the most complex frame of the film.)

To test if frames take longer than the present Adage refresh time, set the button register to 23004 00077, set the switches to 20200 01601, and press PULSE1. (Do not do this while ANTICS is running.) Run the script, and when frames take longer than the refresh time the IC1 light will flash and the display screen will blink in an unmistakable fashion. Repeat steps 1 and 2 to reset this.

If your animation has frames which take longer than 1/30 second, you may try simplifying it by shortening variable names, removing options, etc. You also may

experiment with different camera speeds and refresh rates. Flicker and "strobing" can be monitored by opening the camera's back and looking through the lens while the shutter is operating. (This will work only when the projection speed is roughly the same as the camera speed.) A final alternative is to experiment with the slow-motion feature on the camera while setting the animation speed very low with the #RATE command and the variables FOLLOW-RATE and MOVE-RATE.

Remember that it is impossible to get perfect exposure of the film as long as the longest shutter time (1/40 second) is shorter than the time required to produce an image on the display.

11. When you are finished repeat step 1. Set the switches on the Adage console to 24000 77776 (octal).

9. COMMANDS

NOTATION

Arguments which are enclosed in angle brackets (such as <X>), are optional; they may be omitted. Braces indicate that one alternative should be chosen, as in {ON,OFF}. Elipses indicate that one or more occurrences of the argument may be given, as in (#DISPLAY ATOM1 . . . ATOMn). An underscored parameter is one which will not be evaluated.

Values returned by commands and primitives are meaningless unless specified.

(#START)

#START resets several ANTICS features, #BLANKs the display screen, and initializes the timer used by #END. The following actions are also taken:

1. The stack display is turned off (see #STACK).

2. The animation depth is set to 10000 (see #DEPTH) (#DEPTH = 10000).
3. The animation time delays are reset to normal. The rates of movement of return values and the follower are reset to normal. (Thus the effect of #FAST is reset.)
4. The displaying of EVLIS values is turned on. (EVLIS? = T, EVLIS-MOVE? = T)
5. The movement of return values is turned on. (RETURNS? = T)
6. Prettyprinting is set so as to translate (QUOTE S) to 'S. (SHORTQUOTE = T).
7. The effects of all #BRIEF and #EXCLUDE commands are reset.

(#DEFUN NAME <TYPE> ARGLIST S1 . . . . Sn)

This is identical to LISP/MTS DEFUN. In addition to defining the function, the DEFUN statement itself is prettyprinted at the top of the display screen, and after a short interval, the CONS-cell structure of the resulting LAMBDA-expression is displayed below. A #BLANK command normally should be used immediately before #DEFUN, and a suitable #WAIT and #BLANK should be used after it.

(#STAR FNAME)

This command is used to name the "star" function of an evaluation animation and to prettyprint the function's LAMBDA-Expression at the top of the display screen. It is used after a #BLANK command and before one or more #EVAL or #EVALQ commands. The display will remain until a #BLANK command is given, and only one #STAR command may be given between two #BLANK commands.

(#DISPLAY ATOM1 . . . . ATOMn)

The given atoms' names, along with their values, are displayed in the upper right-hand corner of the screen. The arguments to no-spread LAMBDA-expressions may be displayed by entering numbers in the list of atoms: (#DISPLAY 1 N) will display "ARG 1" and N. Any succeeding changes in the atomic values will be reflected in the display. Whenever ANTICS is waiting (the STEP light button is intensified,) the display will respond to the light pen and values may be changed. The display remains on the screen until the next #BLANK command, and only one #DISPLAY command should be given between two #BLANK commands. Also see the #REDISP command below.

```
(#STACK ATOM1 . . . . ATOMn)
(#STACK #ALL)
(#STACK)
```

This command controls the stack display, which is only apparent during the animation produced by #EVAL or #EVALQ. If a list of atoms is given, only the stack bindings of those atoms will be displayed during animation. The bindings of no-spread LAMBDA's may be shown by entering numbers in the list of atoms. For example, (#STACK N 1) will show the bindings of N and "ARG 1". (#STACK #ALL) causes all bindings to be displayed. In either case, every LAMBDA and PROG will produce at least an empty entry in the stack display (a dashed line). (#STACK) turns off the stack display. The display is also turned off by the #START command.

If the variable ALIST? is non-NIL an a-list is displayed in place of the stack. The #STACK command still must be given and only the specified atoms are shown on the a-list display. When SET or SETQ change a value on the a-list the display also is changed. If SET or SETQ set the value of an atom which is not on the a-list, the value is placed on the end of the a-list, but this is not shown in the a-list display. (This would be an error in LISP 1.5.)

```
(#EVAL FORM <ALIST>)
(#EVALQ FORM <ALIST>)
```

These both cause the evaluation of FORM to be animated. #EVAL first evaluates its argument with LISP/MTS. When the animation is finished, the return value from the evaluation will be left on the screen; it will disappear when the next #EVAL, #EVALQ, or #BLANK is encountered. ALIST may be specified in the form ((ATOM . VALUE) . . . (ATOM . VALUE)). The default ALIST contains T, any values set by #SET or #SETQ, and "top level" values set by SET or SETQ in previous animations.

```
(#BLANK)
```

This clears the display screen, except for the STEP and AUTO light buttons when ANTICS is in STEPMODE. The effects of #STAR and #DISPLAY are reset completely. The #STACK, #BRIEF and #EXCLUDE commands are not reset.

```
(#WAIT SECONDS)
```

ANTICS pauses for the given number of seconds. #WAIT is used in the script to adjust the timing of the movie. When ANTICS is in STEPMODE, #WAIT always will wait until the light pen is used or one of the buttons is used, and it will stop waiting only when STEP or AUTO is selected with the light pen or buttons. See Section 3 for more information.

(#END)

The camera is turned off and the elapsed time from #START is printed.

```
(#SET A1 S1 . . . An Sn)
(#SETQ A1 S1 . . . An Sn)
```

These commands place values permanently on the ANTICS a-list exactly as SET and SETQ normally would be expected to do. (LISP 1.5 generates an error if atoms are set which do not appear on the a-list, while antics places them on the end of the a-list. The result appears exactly like LISP/MTS to the user.) These commands may be used directly in the script or in breakpoints. If the atom whose value is being set is currently displayed on the screen (due to a #DISPLAY command) then the change will be recorded in the display.

```
(SUBROUTINE NAME primitive . . . primitive)
```

The primitives are displayed immediately, with the scale set to 0.5. NAME is ignored. This command is provided primarily for compatibility with GLISP. (See Section 6.)

```
(#DRAW N primitive . . . primitive)
```

The vectors and commands generated by the primitives are displayed on the screen consecutively, at a rate determined by N, so that they appear as though they are being drawn by an invisible hand. If N is zero, the vectors are placed on the screen one at a time as fast as the system permits -- roughly 14 per second. If N is negative, the vectors are placed on the screen |N| at a time at the maximum rate. If N is positive, a pause of N 300ths of a second is taken between displaying each vector.

The scale is set to 0.5, the GLISP default. If an #EVAL or #EVALQ command has been executed, a #BLANK command must be placed before the #DRAW command. This command works well with vectors generated by the GLISP TABLET function.

```
(#PP S <X <Y>>)
```

The LISP expression S is prettyprinted on the display screen at location X,Y. If X or Y are omitted they default to the upper left-hand corner of the display screen. The display will remain on the screen until the next #BLANK command.

```
(#SHOW S <X <Y>>)
```

The CONS-cell structure of S is displayed on the screen at X,Y. S may be a "circular" S-expression. X and Y are the

coordinates of the lower, left-hand corner of the first CONS-cell. If a circular expression is shown, part of the display may be slightly to the left of X (1/4 inch), and care should be taken that this does not extend outside the camera's view. If X or Y are omitted they default to the upper left-hand corner of the display screen. The display will remain on the screen until the next #BLANK command.

(#SHOWFN FNAME <X <Y>>)

The CONS-cell structure of the LAMBDA-expression of the function FNAME is displayed on the screen at X,Y. FNAME is displayed above and intensified. If X or Y are omitted they default to the upper left-hand corner of the display screen. The display will remain on the screen until the next #BLANK command.

(#PLOT)

The contents of the display screen are saved on file -PLOT#. Any number of plots may be saved in this manner, and they may be plotted after leaving LISP by entering the MTS command:

\$RUN PLOT:Q PAR=
BE SURE TO ENTER ONE BLANK AFTER "PAR= ".

(#RATE RATE)

The rate at which evaluation animation proceeds is scaled by RATE. The normal value is 1.0, and higher values produce faster animation. The elapsed time is not precisely proportional to the rate, due to the fixed overhead of CPU time. (For example, an animation of 56.0 seconds duration at (#RATE 1.0) took 30.6 seconds at (#RATE 2.0).) RATE must be greater than zero. The #START command resets the rate to 1.0. #RATE may be used within a breakpoint. See Section 3 for a description of the RATE button.

(#GRID)

This draws a grid of 1-by-1 unit squares on the portion of the screen seen by the camera. This is useful when laying out hand coded displays. The screen is not blanked, and therefore #GRID may be used before or after creating other displays on the screen.

(#FOCUS)

This blanks the screen (exactly as #BLANK does) and then brings up a display which outlines the camera frame area of the display screen and indicates the center of the frame. This is

used to frame and focus the camera. The camera should be adjusted so that the frame outline on the screen is just clearly visible in the camera viewfinder. Press the STEP button (13) to terminate the display and continue with the next command in the script.

(#REDISP ATOM1 . . . . ATOMn)

This command may be used either in the script or in a breakpoint. It assures that the atoms named are displayed correctly if they are currently being displayed by a #DISPLAY command. It is designed to be used after NCONC, RPLACA, or RPLACD, since they can change atomic values without explicitly naming the atoms.

(#ALIAS FNAME ALIAS)

If this command has been given, whenever function FNAME is displayed on the screen, during animation, #SHOWFN, #STAR or #DEFUN, the name ALIAS is used in place of the function's real name. This is useful when, for animation purposes, one wishes to redefine a system function, but the redefinition would disturb the operation of ANTICS. For example to animate EVAL one could define a function \$EVAL and include the command

(#ALIAS \$EVAL EVAL).

To remove the ALIAS from a function, use the command

(#ALIAS FNAME NIL).

The call to the function from #EVAL@ or #EVALQ must use the function's real name, not its alias. #STAR, #SHOWFN, #BRIEF, #EXCLUDE etc. also must use the the function's real name.

There is no corresponding function for atoms since they are kept on the ANTICS EVAL a-list and are completely independent from LISP/MTS.

(#DISP primitive . . . primitive)

#DISP is used to include primitive display commands in the script. The display will remain on the screen until the next #BLANK command unless the FOR or FADE primitives are used.

(#EXIT-DISP SECONDS primitive . . . primitive)

This command is used only to include primitives in the EXIT-FORM portion of an animation breakpoint. The primitives are displayed for the given number of seconds while the rest of animation waits. See Section 6.2 for more information.

(#CAMERA {ON,OFF})

This turns the camera on or off by switching the relay connected to light 1 on the Adage display screen.

(#DEPTH N)

This command sets the maximum number of levels of recursion of ANTICS EVAL that will be displayed on the screen during animation. (The maximum number of EVAL blocks displayed will be N.) When this limit is exceeded, three dots are displayed on the screen to indicate that part of the evaluation is not being animated (though the evaluation does continue.) If a breakpoint sets DISP? = T then animation will continue.

(#BRIEF FNAME1 . . . FNAME_n)
(#BRIEF)

The first form of this command adds the named functions to a list of functions which will be animated as though they were SUBRs: the function call and evaluation of parameters are animated, but the animation of the function's LAMBDA-expression is made completely invisible. The commands (#BRIEF) and (#START) reset the effect of all previous (#BRIEF . . .) commands.

(#EXCLUDE FUNCTION FNAME1 . . . FNAME_n)
(#EXCLUDE ATOM ATOM1 . . . ATOM_n)
(#EXCLUDE FUNCTION)
(#EXCLUDE ATOM)

The first two forms of #EXCLUDE add the specified functions or atoms to respective lists which tell the animation to completely omit the animation of the evaluation of the given functions and atoms. Both EXPRS and SUBRs may be excluded. The second two forms of #EXCLUDE reset the effects of all previous #EXCLUDE commands. #START also resets all #EXCLUDED functions and atoms.

(#STEPMODE)
(#STEPMODE {ON,OFF})

This turns STEPMODE on or off. (#STEPMODE) turns STEPMODE on. See Section 3 for more information.

10. GRAPHICS PRIMITIVES

Graphics primitives may be used in breakpoints or within #DISP or SUBROUTINE commands in the script. For more information see Section 6. Notation is explained at the beginning of Section 9.

(*GOTO X Y)

This moves the display beam to position X,Y on the screen without drawing.

(*PLOT X Y)

This draws a vector from the present position on the screen to X,Y.

(*SYMBOL X Y STRING HEIGHT)

The characters of STRING are displayed on the screen. X,Y is the location of the lower left-hand corner of the first character of STRING, and HEIGHT is the height of the character. (The variable TXTHT may be used for HEIGHT. This will give characters the same size that ANTICS normally uses.) The horizontal spacing of characters is $6/7$ (0.857) times a character's height.

(FOR SECONDS primitive . . . primitive)

The primitives are displayed for the given number of seconds and then turned off. Note that in GLISP the FOR function takes the time parameter in frames, not seconds.

(FADE HOLDTIME FADETIME primitive . . . primitive)

The primitives are displayed for HOLDTIME seconds at normal intensity and then fade out gradually in FADETIME seconds.

(*DASH)

(*UNDASH)

Vectors drawn after (*DASH) are drawn as dashed lines until (*UNDASH) is encountered.

(*INTENSITY N)

N is a number between 0.0 and 1.0 controlling the intensity of the following vectors. Two variables, NORMAL and BRIGHT, are available as standard intensity values (their values are 0.55 and 0.77 respectively.) ANTICS does not reset the intensity if it is changed by the user -- she must be sure to reset it to NORMAL.

```
(GPROG {LABEL,primitive} . . . {LABEL,primitive})
```

This is a PROG type function taken from GLISP. It is used in conjunction with *JUMP. See the GLISP documentation [11] for more information.

```
(*JUMP LABEL)
(*SKIPNZ COUNT)
(*SKIPZ COUNT)
```

These are used in conjunction with GPROG. See the GLISP documentation [11] for more information.

```
(ADJ {DX DY INTENSITY SCALE) primitive . . . . primitive)
```

This is provided primarily for GLISP compatibility. The primitives are displayed at the given intensity, scale, and offsets. Unlike GLISP, dials may not be specified. If an ADJ primitive is nested inside a FADE primitive, the ADJ intensity is ignored. The adjusting values are in the range of -10.0 to 10.0.

```
(SHW1 S X Y)
```

This primitive is the same as the #SHOW command, except that X and Y are required.

```
(PPD FORM X Y WIDTH LEVELS LINE-SKIP)
```

This prettyprints FORM on the display screen. The bottom left-hand corner of the first character is located at X,Y. WIDTH specifies the maximum width of the screen, in characters, that will be used. LEVELS specifies the depth to which FORM will be printed in detail. Sub-expressions of FORM beyond this level will be replaced by "&". LINE-SKIP specifies the distance between successive lines on the screen. If LINE-SKIP is zero only one line will be printed, and if any lines are skipped as a result, three dots will be printed as an indication.

The height of the characters used is determined by the global variable TXTHT.

The global variable SCREEN-WIDTH is set to the width of the screen in characters, and it may be used for WIDTH. TXTHT*2 is set to a good value for LINE-SKIP.

The global variable SHORTQUOTE controls the printing of the QUOTE function. If SHORTQUOTE = T then all QUOTE functions will be abbreviated with "'".

11. USEFUL ANTICS VARIABLES

The following table lists ANTICS variables which the user may wish to access or change. They may be changed either in the script or in breakpoints during animation. The type of numeric variables, integer or real, should not be changed. If a variable is indicated to have dependants then there are other ANTICS variables which are calculated from the particular variable. After changing any of these variables and before executing any ANTICS commands or leaving a breakpoint, the function INITL-CALC should be called to re-calculate the dependant variables. (INITL-CALC takes no parameters.)

VARIABLE NAME	DEFAULT VALUE	SETQ?	DEPEN- DENTS	NOTE	WHERE RESET
BRIGHT	.77	NO	-	1	
NORMAL	.55	NO	-	1	
GLISP-SCALE	.5	YES	NO		
FOLLOW?	T	YES	NO	2	
REFS?	NIL	YES	NO	2	
ROLL-RATE	.75	YES	NO	3	#START, #RATE
FOLLOW-RATE	2.	YES	NO	3	#START, #RATE
MOVE-RATE	.75	YES	NO	3	#START, #RATE
ENTER-WAIT	2.	YES	NO	3	#START, #RATE
EXIT-WAIT	1.	YES	NO	3	#START, #RATE
DEFUN-WAIT	2.	YES	NO	3	#START
SHORT-WAIT	.4	YES	NO	3	#START, #RATE
MEDIUM-WAIT	.8	YES	NO	3	#START, #RATE
DOWN-Y	.6	YES	NO	4	
RIGHT-X	1.	YES	NO	4	
BOXW	.5	YES	YES	4	
ARRL	.2	YES	YES	4	
BARBSIZE	.075	YES	NO	4	
BOXH	.2	YES	YES	4	
EV-*	.05	YES	NO	4	
EV-HT	.1	YES	NO	4	
TXHT	.14	YES	YES	5	
SCREEN-LIMIT	5.	NO	-	6	
ROLL-RATIO	.66	YES	NO	7	
VALUE-LENGTH	24	YES	YES	8	
EVAL-PPLEV	4	YES	NO	9	
VAR-PPLEV	2	YES	NO	9	
STAR-PPLEV	100	YES	NO	9	
FRAME-RATIO	.74	YES	YES	10	
DONT-TOUCH	.1	YES	NO	11	
#DEPTH	10000	YES	NO	12	#START
"*UNDEF* "	"*UNDEF* "	YES	NO	13	
FRAMES/SEC	40.	YES	NO	14	
RETURNS?	T	YES	NO	15	#START
EVLIS?	T	YES	NO	15	#START
EVLIS-MOVE?	T	YES	NO	15	#START
EVLIS-PPLEN	8	YES	NO	16	
SHORTQUOTE	T	YES	NO	17	#START
LOOPS?	T	YES	NO	18	
CONTINUE?	T	YES	NO	18	
SHOWNIL?	NIL	YES	NO	18	
DETAILATOM?	NIL	YES	NO	18	
CONSIZE	.16	YES	YES	19	
ATMHT	.11	YES	NO	19	
YDEC	-.5	YES	NO	19	
MINCONSWIDTH	.55	YES	NO	19	
X-LIMIT	4.5	YES	NO	20	
Y-LIMIT	-4.5	YES	NO	20	
ALIST?	NIL	YES	NO	21	

NOTES:

1. See the primitive *INTENSITY.
2. FOLLOW? is used to turn the follower on or off. REFS? determines whether the follower will move to the variable display when atomic values are referenced.
3. These control the rate at which animation progresses. MOVE-RATE, FOLLOW-RATE and ROLL-RATE control the rates of movement of return values, the follower and rolling of the display. They are in units of screen-units/second.
 ENTER-WAIT and EXIT-WAIT control the time delays before and after entering animation EVAL. SHORT-WAIT and MEDIUM-WAIT control time delays which occur at various times during animation,
 DEFUN-WAIT determines the time delay in #DEFUN between prettyprinting the function and displaying its CONS-cells.
4. These variables determine the dimensions of the EVAL box display during animation. The following figure illustrates their significance. These variables should not be changed during animation.

Figure 4 - Dimensions of an EVAL Box
(actual size)

5. TXTHT is the height of all text used in ANTICS except for atom names in SHOW.
6. SCREEN-LIMIT is the bound for the absolute values of X and Y coordinates. It is provided for reference, and probably should not be changed.
7. ROLL-RATIO determines the fraction of the screen which will disappear when animation rolls part of the display upwards off the screen.
8. VALUE-LENGTH is the approximate maximum number of characters which will be displayed as return values or entries in #DISPLAY or #STACK displays. The actual length displayed depends on the number of vectors required to generate the characters.
9. EVAL-PPLEV is the number of levels of detail that are shown

when prettyprinting forms being evaluated during animation. VAR-PPLEV is the number of levels of detail included during animation when printing return values or entries in the #DISPLAY or #STACK displays. STAR-PPLEV is the number of levels of detail included when prettyprinting for the #STAR or #PP commands.

10. This is the ratio frame-height/frame-width of the camera being used. 0.74 is correct for Super-8 mm. film.
11. DONT-TOUCH is simply a short distance on the screen which may be used when two parts of a display would look best if they were very close but not touching.
12. #DEPTH controls the depth of evaluation. See the command #DEPTH for more information.
13. This is the value given to undefined atoms. Notice the blank added to the name which distinguishes this from the LISP/MTS undefined atom value.
14. FRAMES/SEC should match the refresh rate of the Adage. It is used when producing movement and in the FADE and FOR primitives.
15. RETURNS? determines whether return values will be displayed or not. EVLIS? determines whether values from the evaluation of arguments will be displayed; and EVLIS-MOVE? determines whether they will be moved into a list of argument values.
16. This is the maximum length, in characters, of the EVLIS values that are listed.
17. If SHORTQUOTE = T, (QUOTE S) will be abbreviated 'S whenever it is displayed.
18. These affect #SHOW, #SHOWFN, #DEFUN, and SHW1. LOOPS? determines whether a search will be made for circular S-expressions. If CONTINUE? = T, portions of the display which will not fit on the screen are prettyprinted. (Also see Note 20 below.) If SHOWNIL? = T, all NIL atoms are shown, including those which are CDRs. If DETAILATOM? = T, all atoms are shown to be CONS-cells with their PNAMEs displayed to the right.
19. These are the dimensions used when displaying CONS-cell structures:

Figure 5 - Dimensions of a CONS-cell
(actual size)

20. These are the limits of the screen while displaying CONS-cell structures. When the display attempts to go to the right of or below these limits, it will stop, and depending on the value of CONTINUE?, the remainder will be prettyprinted. (See Note 18 above.)
21. See the command #STACK.

INDEX TO ANTICS USER'S GUIDE

ADJ	62
#ALIAS	59
#BLANK	56
#BRIEF	60
Buttons	42-43
#CAMERA	59
*DASH	61
#DEFUN	45,55
#DEPTH	60
#DISP	46,59
#DISPLAY	40,55
#DRAW	57
#END	57
#EVAL	40,56
#EVALQ	40,56
#EXCLUDE	60
#EXIT-DISP	47,59
FADE	61
#FOCUS	58
FOR	61
*GOTO	61
GPROG	62
#GRID	58
*INTENSITY	61
*JUMP	62
Light Pen	42
Notation	54
#PLOT	58,61
#PP	57
PPD	62
#RATE	43,58
#REDISP	59
#SET	57
#SETQ	57
#SHOW	45,57
#SHOWFN	58
SHW1	62
*SKIPNZ	62
*SKIPZ	62
#STACK	40,56
#STAR	40,55
#START	54
#STEPMODE	60
SUBROUTINE	49,57
*SYMBOL	61
*UNDASH	61
Variables	63
#WAIT	42,56

APPENDIX B - SAMPLE ANTICS SCRIPTS

This script produced the animated film which is included as part of this thesis. It lasts approximately three minutes, twenty seconds, but for educational purposes it probably should be slowed down by a factor of two or three.

```
(PROGN
  (SETQ FRAMES/SEC 30)
  (#FOCUS)
  (#WAIT 10.)
  (#START)
  (#CAMERA ON)
;TITLES
  (#DISP (FADE 6. 2. (*SYMBOL -2.57 1. 'MEMBER 1.)))
  (#WAIT 2.)
  (#DISP (FADE 4. 2. (*SYMBOL -1.8 0. '"BY MARK DIONNE" .3)))
  (#WAIT 6.5)
  (#DISP (FADE 5. 2.
    (*SYMBOL -3. 2.5 '"SUBMITTED IN PARTIAL FULFILLMENT OF" .2)
    (*SYMBOL -2.92 2.0 '"THE REQUIREMENTS FOR THE DEGREE OF" .2)
    (*SYMBOL -1.46 1.5 '"MASTER OF SCIENCE" .2)
    (*SYMBOL -2.57 0. '"UNIVERSITY OF BRITISH COLUMBIA" .2)
    (*SYMBOL -1.11 -.5 '"OCTOBER, 1975" .2)))
  (#WAIT 7.)
  (#BLANK)
  (#RATE .9)
  (SETQ X-LIMIT 5.0)
  (#DEFUN MEMBER(THING LIST)
    (COND ((NULL LIST) NIL)
          ((EQUAL THING (CAR LIST)) LIST)
          (T (MEMBER THING (CDR LIST)))))
  (#WAIT 6.)
  (#BLANK)
  (#STAR MEMBER)
  (#DISPLAY THING LIST)
  (#STACK THING LIST)
;EVALUATION ANIMATION
  (#EVALQ (MEMBER 'A '(C A T)))
  (#WAIT 2.)
  (#BLANK)
; "THE END"
  (#DRAW 0
    (*GOTO 0 0)
    (*GOTO -8.351988 -0.083084)
    (*PLOT -8.151501 -0.183328)
```

(Approximately 100 lines of *PLOT and *GOTO primitives are omitted at this point. They were generated by GLISP using the data tablet and constitute the words "the end" in hand drawn script.)

```
    (*PLOT 5.701635 -1.694323)
    (*PLOT 5.853224 -1.542734))
  (#END) )
```

This script illustrates the display of S-expressions. Figure 1 (page 9) shows the screen after this script has been executed.

```
(PROGN
  (#BLANK)
  (#DISP (*SYMBOL -5. 4.5 '"1 - (SETQ Z '((A) B))" TXTHT))
  (SETQ Z '((A) B))
  (#WAIT 1.)
  (#DISP (*SYMBOL -4.5 4. 'Z= TXTHT))
  (#SHOW Z -4.5 3.5)
  (#WAIT 5.)
  (#DISP (*SYMBOL -5. 1. '"2 - (RPLACD (CDR Z) Z)" TXTHT))
  (NULL (RPLACD (CDR Z) Z))
  (#WAIT 1.)
  (#DISP (*SYMBOL -4.5 .3 'Z= TXTHT))
  (#SHOW Z -4.5 -0.2)
  (#WAIT 5.)
  (#DISP (*SYMBOL 0. 4.5 '"3 - (RPLACA (CDR Z) (CAR Z))" TXTHT))
  (NULL (RPLACA (CDR Z) (CAR Z)))
  (#WAIT 1.)
  (#DISP (*SYMBOL 0.5 4.0 'Z= TXTHT))
  (#SHOW Z 0.5 3.5)
  (#WAIT 5.)
  (#DISP (*SYMBOL 0. 1. '"4 - (RPLACD Z Z)" TXTHT))
  (NULL (RPLACD Z Z))
  (#DISP (*SYMBOL .5 .3 'Z= TXTHT))
  (#SHOW Z 0.5 -.2)
  (#WAIT 5.) )
```

This script animates the execution of a simplified version of EVAL. It uses several of the more advanced features of ANTICS to delete unnecessary detail from the film and to permit the animation of functions with the same names as LISP system functions. The *APPLY function uses user breakpoints in a novel way to access the LISP/MTS APPLY function in order to execute SUBRS. Since the functions animated here are too large to fit on the display screen easily, the viewer should be provided with a copy of EVAL for reference [16].

```
(PROGN

(#START)

(#DISPLAY FORM FN A ARGS)

(#STACK FORM A)

;THE EVAL FUNCTIONS ARE PRINTED AS SKELETONS -- THEY ARE LARGE
(SETQ EVAL-PPLEV 2)

(#EXCLUDE FUNCTION QUOTE CAR CDR NULL)

(#EXCLUDE ATOM FORM FN A EXP ARGS C M NIL T J)

(#BRIEF !GET SASSOC APPLY* *PAIR)

(#ALIAS !GET GET)

;*PAIR IS PART OF ANTICS!
(#ALIAS *PAIR PAIR)

(#ALIAS !EVAL EVAL)

(#ALIAS !EVCON EVCON)

(#ALIAS !EVLIS EVLIS)

(#ALIAS !APPLY APPLY)

(#SETQ A '((T . T)))

(DEFUN !APPLY (FN ARGS A)
  (COND ((NULL FN) NIL)
        ((ATOM FN)
         (COND ((!GET FN 'EXPR) (!APPLY EXP ARGS A))
               ((!GET FN 'SUBR) (APPLY* EXP ARGS))
               (T (!APPLY (SASSOC FN A) ARGS A))))
        ((EQ (CAR FN) 'LAMBDA)
         (!EVAL (CADDR FN)
                  (NCONC (*PAIR (CADR FN) ARGS) A)))
        (T (!APPLY (!EVAL FN A) ARGS A))))

(DEFUN !EVAL (FORM A)
  (COND ((NULL FORM) NIL)
        ((NUMBERP FORM) FORM)
```

```

((ATOM FORM) (SASSOC FORM A))
((EQ (CAR FORM) 'QUOTE) (CADR FORM))
((EQ (CAR FORM) 'COND) (!EVCON (CDR FORM) A))
((ATOM (CAR FORM))
  (COND ((!GET (CAR FORM) 'EXPR)
    (!APPLY EXP (!EVLIS (CDR FORM) A) A))
    ((!GET (CAR FORM) 'SUBR)
    (APPLY* (CAR FORM) (!EVLIS (CDR FORM) A)))
    (T (!EVAL (CONS (SASSOC (CAR FORM) A)
      (CDR FORM))
      A))))
  (T (!APPLY (CAR FORM) (!EVLIS (CDR FORM) A) A))))

(DEFUN !EVCON (C A)
  (COND ((!EVAL (CAAR C) A) (!EVAL (CADAR C) A))
    (T (!EVCON (CDR C) A))))

(DEFUN !EVLIS (M A)
  (# (AND DISP? (SETQ DISP? (LESSP #TIMES 2)))
    (COND ((NULL M) NIL)
      (T (CONS (!EVAL (CAR M) A) (!EVLIS (CDR M) A))))))

(DEFUN !GET (FN TYPE)
  (SETQ EXP (GET FN TYPE))
  (COND ((NULL EXP) NIL)
    ((EQ TYPE 'SUBR) '*)
    (EXP)))

(DEFUN SASSOC (AT LIST)
  (CDR (ASSOC AT LIST '(*UNDEFINED*))))

(DEFUN APPLY* (EXP ARGS)
  (# (APPLY (*VALUE 'EXP A NIL) (*VALUE 'ARGS A NIL))))

;AN EXAMPLE
(DEFUN FOO (X) (CONS X X))

(#EVALQ (!EVAL ' (FOO 'Z) A))

```

APPENDIX C - ANTICS PROGRAM LISTING

```

;*****
;   LISP BASIC
;*****
(STATUS (32 6))

```

```

(MTS "SET LIBSRCH=GRAF:AGTBASIC+AGT:BASIC")

```

```

;PARTS OF LISP BASIC WHICH ARE NOT USED ARE OMITTED
(DEFINE (DUMMY1 SUBR (0 "GRAF:LISP BASIC.O" AGPLOT))
        (#DUMP# SUBR (3 "GRAF:BDUMP.O" BDUMP))
        (GETSPACE SUBR (0 "GRAF:LISP BASIC.O" LGETSPA))
        (AGTRD SUBR (0 "GRAF:LISP BASIC.O" LAGTRD))
        (BUTTON SUBR (0 "GRAF:LISP BASIC.O" LBUTTON))
        (DIAL SUBR (0 "GRAF:LISP BASIC.O" LDIAL))
        (PENHIT SUBR (0 "GRAF:LISP BASIC.O" LPENHIT))
        (INDEX SUBR (0 "GRAF:LISP BASIC.O" LINDEX))
        (AGTBUF SUBR (0 "GRAF:LISP BASIC.O" LAGTBUF))
        (AGTCON SUBR (0 "GRAF:LISP BASIC.O" LAGTCON))
        (CONAGT SUBR (0 "GRAF:LISP BASIC.O" LCONAGT))
        (AGTCVT SUBR (0 "GRAF:LISP BASIC.O" LAGTCVT))
        (AGTDSP SUBR (0 "GRAF:LISP BASIC.O" LAGTDSP))
        (AGTEXT SUBR (0 "GRAF:LISP BASIC.O" LAGTEXT))
        (AGPLOT SUBR (0 "GRAF:LISP BASIC.O" LAGPLOT)))

```

```

(SETQ DASH 1
      UNDASH 2
      SCALE-D 4
      SCALE 5
      DX-D 6
      DX 7
      DY-D 8
      DY 9
      INT-D 10
      INT 11
      PENON 15
      PENOFF 16
      MOVE 19
      MOVER 20
      JUMP 21
      JUMPR 22
      SKIPNZ 23
      SKIPZ 24
      INCR-D 25
      INCR 26
      LIGHTSON 34
      LIGHTSOFF 36)

```

```

(DEFUN BDUMP (ARRAY START END)
  (PROG NIL
    (OR (AND (GREATERP START 0)
              (LESSP START (ADD1 END)))

```



```

      (LESSP END 6001))
    (RETURN (NULL (PRINT '"BAD INDEX PASSED TO
                      BDUMP"))))
  (#DUMP# ARRAY START END))

```

```

;*****
;  INITIALIZATION ETC.
;*****
; IFADG CHECKS TO SEE IF WE ARE AT THE ADAGE TERMINAL
(DEFINE (RTWAIT SUBR (3 *LIBRARY RTWAIT))
  (AGT? SUBR (0 "MASD:AGTEST.O" IFADG))
  (STOP BUG
    ((LAMBDA NIL
      (AND PLOTTED?
        (PRINT '"PLOT OUTPUT IS ON FILE
          -PLOT#"))))))

```

```

; FIX SYSTEM FUNCTION PLEN SO THAT IT WORKS FOR NUMERIC ATOMS
(PUT '#PLEN# 'SUBR (GET 'PLEN 'SUBR))

```

```

(PUT 'PLEN
  'EXPR
  '(LAMBDA (ATOM)
    (COND ((NUMBERP ATOM)
      (#PLEN# (MKATOM ATOM)))
      ((#PLEN# ATOM)))))

```

```

; INITIALIZE "ONE-TIME" STUFF
; THESE SHOULD GENERALLY NOT BE CHANGED BY THE USER
(SETQ ATAGT? (AGT?)
  STAR NIL
  ?STEPMODE NIL
  PLOTTED? NIL
  STAR-FLAG NIL
  FADE? NIL
  SUBROUTINE? NIL
  EVALING? NIL
  ON 'ON
  OFF 'OFF
  SCREEN-LIMIT 5.
  CAMERA? NIL
  TIME-FILM-STARTED NIL
  RESTART-CAMERA NIL
  USER-FORM '#
  STAR-FORM '*'
  FULLMODE 2
  PAREN-LEN 4
  "(" " " ("
  ")" " " ") "
  "." " " "."
  "" "" ""
  & '8
  EVAL-COUNTER 1

```

```

RATE-BUTTON 14
QUIT-BUTTON 10
STEP-BUTTON 13
BACKUP-BUTTON 5
STEPMODE-BUTTON 15
AUTO-BUTTON 16
PLOT-BUTTON 9
BREAK-BUTTON 11
CAMERA-BUTTON 12
NORMAL-BUTTONS (LIST STEP-BUTTON
                  QUIT-BUTTON
                  AUTO-BUTTON
                  STEPMODE-BUTTON
                  PLOT-BUTTON
                  BREAK-BUTTON
                  RATE-BUTTON
                  CAMERA-BUTTON)
EVAL-BUTTONS (CONS BACKUP-BUTTON NORMAL-BUTTONS))

```

```

;IF THE FOLLOWING ARE CHANGED EVERYTHING MUST BE RELOADED
(SETQ LWA-ADAGE 6000
      FOLLOWER-SIZE 0.1
      *SCALE 1.
      BRIGHT 0.77
      NORMAL 0.55)

```

```

;THESE MAY BE CHANGED BY THE USER
(SETQ GLISP-SCALE 0.5
      FOLLOW? T
      ALIST? NIL
      REFS? NIL
      "*UNDEF* " ""*UNDEF* "
      DEFUN-WAIT 2.
      FRAMES/SEC 40.
      FILM-USED 0.
      FILM-LENGTH NIL
      TIMESCALE 1.
      FRAME-RATIO 0.74
      TXTHT 0.14
      DONT-TOUCH 0.1
      ITSY-BITSY 0.05
      DOWN-Y 0.6
      RIGHT-X 1.
      EVLIS-PPLEN 8
      BOXW 0.5
      ARRL 0.2
      BARBSIZE 0.075
      BOXH 0.2
      EV-* 0.05
      EV-HT 0.1
      ROLL-RATIO 0.66
      VALUE-LENGTH 24
      STAR-PPLEV 100
      EVAL-PPLEV 4
      VAR-PPLEV 2)

```

```

;THESE ARE USED BY #SHOW
;CONSIZE IS THE HEIGHT OF A CONS CELL
(SETQ YDEC -0.5
  ATMHT 0.11
  CONSIZE 0.16
  SOMESPACE 0.035
  SHOW-BARBSIZE 0.05
  MINCONSWIDTH 0.55
  X-LIMIT 4.5
  Y-LIMIT -4.5
  SHOWNIL? NIL
  DETAILATOM? NIL
  LOOPS? T
  CONTINUE? T)

```

```

;RE-CALCULATE DEPENDANT CONSTANTS
(DEFUN INITL-CALC ()
  (SETQ DY-DX-SCALE (TIMES *SCALE 0.1)
    LINE-SKIP (TIMES TXTHT 1.5)
    NEG-SCREEN-LIMIT (MINUS SCREEN-LIMIT)
    BOTTOM-OF-FILM (SUB SCREEN-LIMIT
      (TIMES FRAME-RATIO
        (ADD SCREEN-LIMIT
          SCREEN-LIMIT)))

    TXTHT*2 (ADD TXTHT TXTHT)
    TXTHT/2 (TIMES TXTHT 0.5)
    BOXW/2 (TIMES BOXW 0.5)
    BOXH/2 (TIMES BOXH 0.5)
    EVAL-BLK-HT (ADD ARRL BOXH)
    AGT-VALUE-LENGTH (TIMES VALUE-LENGTH 5)
    LOOPDEC (TIMES 0.5 (ADD YDEC CONSIZE))
    CONSIZE/2 (TIMES CONSIZE 0.5)
    CONSIZE*2 (ADD CONSIZE CONSIZE)
    VARS-X (SUB SCREEN-LIMIT
      (TIMES VALUE-LENGTH TXTHT))
    STACK-X VARS-X
    SCREEN-WIDTH (FIX (DIVIDE (SUB SCREEN-LIMIT
      NEG-SCREEN-LIMIT)
      TXTHT))))

```

```

(INITL-CALC)

```

```

;SET SPEED OF ANIMATION
(DEFUN #RATE (N)
  (SETQ ENTER-WAIT (DIVIDE 2. N)
    EXIT-WAIT (DIVIDE 1. N)
    MOVE-RATE (TIMES 0.75 (MAX (FLOAT N) 1.))
    ROLL-RATE (MIN 40.
      (TIMES 0.75 (MAX (FLOAT N) 1.)))
    FOLLOW-RATE (TIMES 2. (MAX (FLOAT N) 1.))
    SHORT-WAIT (DIVIDE 0.4 N)
    MEDIUM-WAIT (DIVIDE 0.8 N)))

```

```

;SET UP AGT ARRAYS
;(ALIST-LPAREN, ABOUT LINE --65, IS SPACE FOR THE ALIST OUTER
;
(COND (ATAGT? (SETQ :ARRAY (GETSPACE LWA-ADAGE)
                    :FOLLOW (GETSPACE 15)
                    :ROLL (GETSPACE 8)
                    :CONTROL (GETSPACE 124)
                    ROLL-LEN 7
                    ROLL-EOL 8
                    :EOL (GETSPACE 1)
                    :NOP (GETSPACE 1)
                    :TEMP (GETSPACE 10)
                    :BRIGHT (GETSPACE 1)
                    :NORMAL (GETSPACE 1)
                    :STACK (GETSPACE 1)
                    :SKIP-VAL (GETSPACE 1))
      (INDEX :ROLL 1)
      (SETQ ROLL-SKIP (INDEX :ROLL))
      (AGTCON SKIPZ 0 0 :ROLL)
      (SETQ ROLL-INCR (INDEX :ROLL))
      (AGTCON INCR 0 0 :ROLL)
      (SETQ ROLL-DY (INDEX :ROLL))
      (AGTCON DY 0 0 :ROLL)
      (AGTCON INT 0 NORMAL :ROLL)
      (AGTCON DX 0 0 :ROLL)
      (AGTCON PENON 0 0 :ROLL)
      (SETQ ROLL-JUMP (INDEX :ROLL))
      (AGTCVT 0 0 NIL NIL :ROLL)
      (AGTCVT 0 0 NIL T :ROLL)
      (AGTCON INT 0 BRIGHT :BRIGHT 1)
      (AGTCON INT 0 NORMAL :NORMAL 1)
      (AGTCON SKIPZ 0 0 :FOLLOW 2)
      (AGTCON INCR 0 0 :FOLLOW)
      (AGTCON DX 0 0 :FOLLOW)
      (AGTCON SKIPZ 0 0 :FOLLOW)
      (AGTCON INCR 0 0 :FOLLOW)
      (AGTCON DY 0 0 :FOLLOW)
      (AGTCVT 0.
        FOLLOWER-SIZE
        NIL
        NIL
        :FOLLOW)
      (AGTCVT (TIMES 0.6 FOLLOWER-SIZE)
        (TIMES -0.8 FOLLOWER-SIZE)
        T
        NIL
        :FOLLOW)
      (AGTCVT (TIMES -0.95 FOLLOWER-SIZE)
        (TIMES 0.3 FOLLOWER-SIZE)
        T
        NIL
        :FOLLOW)
      (AGTCVT (TIMES 0.95 FOLLOWER-SIZE)
        (TIMES 0.3 FOLLOWER-SIZE)
        T

```

```

NIL
:FOLLOW)
(AGTCVT (TIMES -0.6 FOLLOWER-SIZE)
(TIMES -0.8 FOLLOWER-SIZE)
T
NIL
:FOLLOW)
(AGTCVT 0. FOLLOWER-SIZE T NIL :FOLLOW)
(SETQ FOLLOW-LENGTH (SUB1 (INDEX :FOLLOW)))
(AGTCON JUMPR FOLLOW-LENGTH 0 :FOLLOW 1)
(AGTCON SCALE 0 *SCALE :CONTROL 1)
(SETQ CAMERA-CONTROL (INDEX :CONTROL))
(AGTCON LIGHTSOFF 1 0 :CONTROL)
(AGTCON PENOFF 0 0 :CONTROL)
(AGTEXT -0.72 -4. 0.28 'ANTICS :CONTROL)
(INDEX :CONTROL
(ADD1 (SETQ STEP-CONTROL (INDEX
:CONTROL))))
(AGTCON PENON 0 0 :CONTROL)
(SETQ INTS-STEP (INDEX :CONTROL))
(AGTCON INT 0 NORMAL :CONTROL)
(AGTEXT NEG-SCREEN-LIMIT
NEG-SCREEN-LIMIT
TXTHT
'STEP
:CONTROL)
(AGTCON INT 0 NORMAL :CONTROL)
(SETQ END-STEP (INDEX :CONTROL))
(AGTEXT (ADD NEG-SCREEN-LIMIT
(TIMES TXTHT 6))
NEG-SCREEN-LIMIT
TXTHT
'AUTO
:CONTROL)
(AGTCON PENOFF 0 0 :CONTROL)
(SETQ END-AUTO (INDEX :CONTROL))
(INDEX :CONTROL (ADD1 END-AUTO))
(SETQ EVAL-RETURN (INDEX :CONTROL)
STACK-ROLL-ADDR EVAL-RETURN)
(AGTCON SKIPZ 0 0 :CONTROL)
(AGTCON INCR 0 0 :CONTROL)
(AGTCON DY 0 0 :CONTROL)
(AGTCON PENOFF 0 0 :CONTROL)
(SETQ ALIST-LPAREN (INDEX :CONTROL))
(REPEAT '(AGTCVT 0 0 NIL NIL :CONTROL)
PAREN-LEN)
(SETQ STACK-JUMP (INDEX :CONTROL))
(AGTCON JUMP LWA-ADAGE 0 :CONTROL)
(SETQ CONTROL-LEN (INDEX :CONTROL)
INIT-FREE CONTROL-LEN)
(AGTCVT 0 0 NIL T :CONTROL)
(AGTCON JUMP
INIT-FREE
0
:CONTROL
STEP-CONTROL)
(AGTCON JUMP

```

```

INIT-FREE
0
:CONTROL
END-AUTO)
(AGTCON JUMP EVAL-RETURN 0 :STACK 1)
(AGTCON JUMPR
AGT-VALUE-LENGTH
0
:SKIP-VAL
1)
(AGTCVT 0 0 NIL NIL :NOP 1)
(AGTCVT 0 0 NIL T :EOL 1)))

```

```

;*****
; FUNCTIONS CALLED BY A USER
;*****
;SET ANIMATION VARIABLES
(DEFUN #SETQ FEXPR (Q)
  (COND ((LISTP (CAR Q))
    (ERRMSG ATTEMPT TO SETQ: ((CAR Q))))
    ((NULL (CDDR Q))
    (*SETQ1 (CAR Q)
      (EVAL (CADR Q))
      (COND (EVALING? A)
        (*ALIST))))
    (T (*SETQ1 (CAR Q)
      (EVAL (CADR Q))
      (COND (EVALING? A)
        (*ALIST)))
      (APPLY '#SETQ (CDDR Q))))))

```

```

(DEFUN #SET FEXPR (Q)
  (PROG (AT)
    (SETQ AT (EVAL (CAR Q)))
    (COND ((LISTP AT)
      (ERRMSG ATTEMPT TO SET: (AT)))
      ((NULL (CDDR Q))
      (*SETQ1 AT
        (EVAL (CADR Q))
        (COND (EVALING? A)
          (*ALIST))))
      (T (*SETQ1 AT
        (EVAL (CADR Q))
        (COND (EVALING? A)
          (*ALIST)))
        (APPLY '#SET (CDDR Q))))))

```

```

;DEFINE AND DISPLAY A FUNCTION
(DEFUN #DEFUN FEXPR (F)
  (PROG (NLINES)
    (APPLY 'DEFUN F)
    (INDEX :ARRAY 1)
    (SETQ NLINES (FIX (PPD (CONS 'DEFUN F)
      NEG-SCREEN-LIMIT

```

```

                                (SUB SCREEN-LIMIT
                                  TXTHT)
                                SCREEN-WIDTH
                                100
                                TXTHT*2)))

(DISPL)
(#WAIT DEFUN-WAIT)
(APPLY1 '#SHOWFN
        (CAR F)
        NEG-SCREEN-LIMIT
        (SUB SCREEN-LIMIT
          (TIMES TXTHT*2 (ADD 2 NLINES))))))

;PRETTYPRINT AN S-EXPRESSION AT X Y ON THE SCREEN
(DEFUN #PP N
  (INDEX :ARRAY 1)
  (PROG (X)
    (PPD (ARG 1)
      (SETQ X (COND ((GREATERP N 1) (ARG 2))
                    ((ADD NEG-SCREEN-LIMIT
                        CONSIZE))))
      (COND ((GREATERP N 2) (ARG 3))
            ((SUB SCREEN-LIMIT CONSIZE*2)))
      (FIX (DIVIDE (SUB SCREEN-WIDTH X) TXTHT))
      STAR-PPLEV
      TXTHT*2))
  (DISPL))

;(#DISPLAY VAR VAR ...)
;CONTROLS DISPLAY OF VARIABLES.
;IF VAR IS A NUMBER, IT CONTROLS (ARG N)
(DEFUN #DISPLAY NEXPR N
  (PROG (M Y A ADDR)
    (INDEX :ARRAY 1)
    (AGTCON PENON 0 0 :ARRAY)
    (SETQ M 1
          Y TOP-SCREEN-AGT
          DISP-VARS NIL
          RMARGIN (MIN RMARGIN
                      (SUB VARS-X DONT-TOUCH)))
    LOOP
      (SETQ A (ARG M))
      (AND (NUMBERP A) (SETQ A (MKATOM '"ARG " A)))
      (SETQ DISP-VARS (CONS A DISP-VARS))
      (PUT A 'BEGIN (INDEX :ARRAY))
      (*INTENSITY NORMAL)
      (AGTEXT VARS-X
              Y
              TXTHT
              (MKATOM A '" =')
              :ARRAY)
      (PUT A 'VAL-ADDR (SETQ ADDR (INDEX :ARRAY)))
      (PUT A 'Y Y)
      (PPD (PUT A
                'VALUE

```

```

      (*VALUE A *ALIST '"*UNDEF* ")
(PUT A
 'X
      (ADD VARS-X
        (TIMES TXTHT (ADD (PLEN A) 3)))
Y
VALUE-LENGTH
VAR-PPLEV
0)
(SETQ A (SUB (INDEX :ARRAY) ADDR)
Y (SUB Y TXTHT*2))
(AND (LESSP A AGT-VALUE-LENGTH)
      (AGTCON JUMPR
        (SUB AGT-VALUE-LENGTH A)
        0
        :ARRAY))
(INDEX :ARRAY (ADD ADDR AGT-VALUE-LENGTH))
(SETQ M (ADD1 M))
(OR (GREATERP M N) (GO LOOP))
(*INTENSITY NORMAL)
(AGTCON PENOFF 0 0 :ARRAY)
(*DASH)
(*GOTO SCREEN-LIMIT (ADD TXTHT Y))
(*PLOT RMARGIN (ADD TXTHT Y))
(*PLOT RMARGIN
  (MIN SCREEN-LIMIT
    (ADD TXTHT*2 TOP-SCREEN-AGT)))
(*UNDASH)
(SETQ STACK-TOP Y
Y-FOLLOW-VARS Y
X-FOLLOW-VARS (SUB VARS-X
  (ADD FOLLOWER-SIZE
    DONT-TOUCH))
FWA-VARS (DISPL)))

```

```

;A USER CALL TO MAKE SURE AN ATOMIC VALUE IS DISPLAYED
;
; PROPERLY.
; IT IS USED AFTER AN NCONC ETC. WHEN A VALUE CHANGE IS NOT
; APPARENT TO THE ANIMATION.

```

```

(DEFUN #REDISP NEXPR (AT)
  (VAR-CHANGE AT
    (*VALUE AT
      (COND (EVALING? A)
        (*ALIST))
      '"*UNDEF* ")))

```

```

;NAME THE FUNCTION WHICH IS THE STAR OF ANIMATION

```

```

(DEFUN #STAR NEXPR (F)
  (PROG (N)
    (PUT F
      'STAR
      (COPISTAR (GET F
        'EXPR
        '(RETURN (ERRMSG (F)
          IS

```



```

NOT
AN
"EXPR.")))))

```

```

(SETQ STAR F)
(INDEX :ARRAY 1)
(*INTENSITY BRIGHT)
(AGTCVT 0 0 NIL NIL :ARRAY)
(AGTEXT NEG-SCREEN-LIMIT
TOP-SCREEN-AGT
TXTHT
(MKATOM (FNAME F) ':)
:ARRAY)
(*INTENSITY NORMAL)
(SETQ STAR-FLAG T
N (PPD (GET F 'STAR)
NEG-SCREEN-LIMIT
(SUB TOP-SCREEN-AGT TXTHT*2)
(FIX (DIVIDE (ADD RMARGIN
SCREEN-LIMIT)
TXTHT))
STAR-PPLEV
TXTHT*2)
STAR-FLAG NIL
TOP-SCREEN-AGT (SUB TOP-SCREEN-AGT
(TIMES (ADD 1 N)
TXTHT*2)))
(*INTENSITY NORMAL)
(*DASH)
(*GOTO NEG-SCREEN-LIMIT TOP-SCREEN-AGT)
(*PLOT RMARGIN TOP-SCREEN-AGT)
(*UNDASH)
(SETQ TOP-SCREEN-AGT (SUB TOP-SCREEN-AGT
TXTHT*2)
FWA-STAR (DISPL))
(OR DISP-VARS (SETQ STACK-TOP TOP-SCREEN-AGT)))

```

;COPY THE STAR FUNCTION INSERTING BREAKPOINTS FOR THE ADDRESSES
; TO INTENSIFY.

```

(DEFUN COPYSTAR (FORM)
(COND ((ATOM FORM) FORM)
((ATOM (CAR FORM))
(COND ((EQNAME (CAR FORM) USER-FORM)
(CONS (CAR FORM)
(CONS (CADR FORM)
(AND (CDDR FORM)
(CONS (COPYSTAR (CADDR FORM))
(CDDDR FORM))))))
(T (LIST STAR-FORM
NIL
(COND ((MEMQ (CAR FORM)
'LAMBDA NLAMBDA
FLAMBDA
PROG))
(CONS (CAR FORM)
(CONS (CADR FORM)
(MAPCAR 'COPYSTAR

```

```

(CDDR
FORM))))))
  ((EQ (CAR FORM) 'COND)
    (CONS (CAR FORM)
      (MAPCAR '(LAMBDA (X)
        (MAPCAR
          'COPYSTAR
            X))
        (CDR FORM))))))
    ((EQ (CAR FORM) 'SETQ)
      (PROG (N)
        (CONS (CAR FORM)
          (MAPCAR '(LAMBDA
            (X)
              (COND (N (SETQ N NIL)
                (COPYSTAR X))
                (T (SETQ N T)
                  X))))
            (CDR
              FORM))))))
      ((EQ (CAR FORM)
        'SELECT)
        (CONS (CAR FORM)
          (CONS (COPYSTAR (CADR
            FORM))
              (SELCOPY (CDDR
                FORM))))))
        ((EQ (CAR FORM)
          'SELECTQ)
          (CONS (CAR FORM)
            (CONS (COPYSTAR (CADR
              FORM))
              (SELQCOPY (CDDR
                FORM))))))
          (T (CONS (CAR FORM)
            (MAPCAR 'COPYSTAR
              (CDR
                FORM))))))
          (T (MAPCAR 'COPYSTAR FORM)))

```

```

(DEFUN SELQCOPY (F)
  (COND ((NULL (CDR F)) (LIST (COPYSTAR (CAR F)))))
  (T (CONS (CONS (CAAR F)
    (MAPCAR 'COPYSTAR (CDAR F)))
    (SELQCOPY (CDR F))))))

```

```

(DEFUN SELCOPY (F)
  (COND ((NULL (CDR F)) (LIST (COPYSTAR (CAR F)))))
  (T (CONS (MAPCAR 'COPYSTAR (CAR F))
    (SELCOPY (CDR F))))))

```

```

;(#STACK VAR1 VAR2 ... <#ALL>)
(DEFUN #STACK FEXPR (L)
  (COND ((NULL (CAR L)) (SETQ STACK? NIL))
        (T (MAP '(LAMBDA (L)
                  (AND (NUMBERP (CAR L))
                      (RPLACA L
                        (MKATOM "ARG "
                          (CAR L))))))
            L)
    (SETQ STACK-VARS L STACK? T)
    (AND (MEMQ '#ALL STACK-VARS)
         (SETQ STACK-VARS T))))

;MAKE FUNCTIONS AND ATOMS INVISIBLE
(DEFUN #EXCLUDE FEXPR (L)
  (COND ((EQ (CAR L) 'ATOM)
        (COND ((CDR L)
                (SETQ EXCLUDE-ATOMS (NCONC (CDR L)
                                              EXCLUDE-ATOMS)))
          (T (SETQ EXCLUDE-ATOMS NIL))))
        ((EQ (CAR L) 'FUNCTION)
         (COND ((CDR L)
                (SETQ EXCLUDE-FNS (NCONC (CDR L)
                                           EXCLUDE-FNS)))
          (T (SETQ EXCLUDE-FNS NIL))))))

;MAKE FUNCTIONS LOOK LIKE SUBRS
(DEFUN #BRIEF FEXPR L
  (COND ((NULL L) (SETQ BRIEF-FNS NIL))
        ((SETQ BRIEF-FNS (NCONC (ARG 1) BRIEF-FNS))))

;THE VIEWER WILL BE ACQUAINTED WITH A FUNCTION BY ITS ALIAS,
;NOT ITS REAL NAME
(DEFUN #ALIAS NEXPR (FN ALIAS)
  (PUT FN 'ALIAS ALIAS))

;RETURN ALIAS IF IT EXISTS, OTHERWISE REAL NAME
(DEFUN FNAME (FN)
  (OR (GET FN 'ALIAS) FN))

;WAIT A GIVEN NUMBER OF SECONDS (CALLED BY *EVAL)
(DEFUN #WAIT (SECS)
  (COND (?STEPMODE (AGTDSP INTS-STEP
                          NIL
                          FULLMODE
                          1
                          :BRIGHT)
        (AGTRD 600)
        (AGTDSP INTS-STEP
                  NIL

```

```

                                FULLMODE
                                1
                                :NORMAL)
                                (STEP-AGT))
(T (AND FILM-LENGTH
    CAMERA?
    TIME-FILM-STARTED
    (GREATERP (ADD (TIMES SECS TIMESCALE)
                   FILM-USED
                   (TIMES 9.999999E-4
                     (SUB (STATUS 39)
                          TIME-FILM-STARTED))))
    FILM-LENGTH)
  (PROGN (#CAMERA NIL)
    (MSSG INSERT
      FILM
      THEN
      TYPE
      NIL)
    (SETQ FILM-USED 0.)
    (BREAK)
    (#CAMERA T)))
  (AGTRD (TIMES TIMESCALE SECS))
  (STEP-AGT)))

;WAIT FOR A FEW SECONDS
(DEFUN REALWAIT (SECS)
  (SETQ SECS (TIMES SECS TIMESCALE))
  (REPEAT (RTWAIT 18000) (FIX (DIVIDE SECS 60.)))
  (RTWAIT (REMAIN (FIX (TIMES SECS 300)) 18000)))

;PROCESS BUTTON OR PEN HIT
(DEFUN STEP-AGT ()
  (SELECT (BUTTON)
    (0 (COND ((EQ 0 (PENHIT))) ((EVAL-PEN))))
    (BACKUP-BUTTON (AND EVALING?
                        (PROGN (AGTDSP (CADR LOCLIST)
                                     NIL
                                     FULLMODE
                                     1
                                     :STACK)
                              (WIPE-VAL)
                              (SETQ STACK-BACKUP STACK?)
                              (*UNEVAL* 2))))))
    (QUIT-BUTTON (SETQ EVALING? NIL)
                  (AND CAMERA? (#CAMERA NIL))
                  (UNEVAL -1 '***STOPPED***))
    (STEPMODE-BUTTON (#STEPMODE T)
                     (REALWAIT SHORT-WAIT)
                     (#WAIT 1))
    (AUTO-BUTTON (#STEPMODE NIL))
    (PLOT-BUTTON (#PLOT) (#WAIT 1.))
    (BREAK-BUTTON (BREAK) (#WAIT 0))
    (RATE-BUTTON (SETQ #RATE (TIMES 2.
                                (ADD 1.1

```

```

(DIAL 1)))
(PRINT (LIST '#RATE #RATE))
(#RATE #RATE)
(REALWAIT MEDIUM-WAIT)
(#WAIT 0))
(CAMERA-BUTTON (#CAMERA (NOT CAMERA?))
  (REALWAIT SHORT-WAIT)
  (#WAIT 0))
NIL))

```

```

;PROCESS PEN HIT
(DEFUN EVAL-PEN ()
  (PROG (HIT)
    (SETQ HIT (PENHIT))
    (COND ((LESSP HIT END-STEP)
      (REALWAIT SHORT-WAIT))
      ((LESSP HIT END-AUTO) (#STEPMODE NIL))
      ((LESSP HIT FWA-EVAL)
        (#STEPMODE T)
        (CHANGE-VAR)
        (#WAIT 1))
      (T (PROG (N LIST)
        (SETQ LIST LOCLIST N 0)
        LOOP
          (COND ((NOT (NUMBERP (CAR LIST)))
            (RETURN NIL))
            ((GREATERP HIT
              (CAR LIST)))
            (T (SETQ LIST (CDR LIST)
              N (ADD1 N))
              (GO LOOP)))
          (AGTDSP (CAR LIST)
            NIL
            FULLMODE
            1
            :STACK)
          (WIPE-VAL)
          (REALWAIT SHORT-WAIT)
          (SETQ STACK-BACKUP STACK?)
          (#STEPMODE T)
          (*UNEVAL* N))))))

```

```

;BACK-UP ANIMATION N LEVELS
(DEFUN *UNEVAL* (##N##)
  (COND ((OR (LESSP ##N## 2) (NOT (UNEVAL 'EVAL1)))
    (INDEX :ARRAY 1)
    (UNEVAL '*EVAL T))
  (T (UNEVAL 'EVAL1
    (LIST '*UNEVAL* (SUB1 ##N##))))))

```

```

;PROCESS A PEN HIT OF A VARIABLE
(DEFUN CHANGE-VAR ()
  (PROG (HIT LIST AT)
    (SETQ LIST DISP-VARS

```

```

      HIT (SUB (PENHIT) FWA-VARS))
LOOP
  (OR (LESSP HIT (GET (CAR LIST) 'BEGIN))
      (GREATERP HIT
        (ADD AGT-VALUE-LENGTH
          (GET (CAR LIST) 'VAL-ADDR)))
      (GO GOTIT))
  (SETQ LIST (CDR LIST))
  (OR (NULL LIST) (GO LOOP))
  (RETURN NIL)
GOTIT
  (SETQ AT (CAR LIST))
  (AGTDSP (SETQ HIT (ADD FWA-VARS
    (GET AT 'BEGIN)
    -1))
    NIL
    FULLMODE
    1
    :BRIGHT)
  (MSSG TYPE THE NEW VALUE FOR: (AT))
  (*SETQ1 AT
    (READ ERRIN)
    (COND (EVALING? A)
      (*ALIST)))
  (REALWAIT MEDIUM-WAIT)
  (AGTDSP HIT NIL FULLMODE 1 :NORMAL))

; START A NEW SEQUENCE--INITIALIZE EVERYTHING, WIPE SCREENS,
;                                     TIME=0
(DEFUN #START ()
  (#RATE 1.)
  (SETQ STACK? NIL
    RETURNS? T
    EVLIS-MOVE? T
    EVLIS? T
    SHORTQUOTE T
    #DEPTH 10000
    STACK-VARS NIL
    EXCLUDE-ATOMS NIL
    EXCLUDE-FNS NIL
    BRIEF-FNS NIL)
  (AND ATAGT? (AGTRD 0 NORMAL-BUTTONS))
  (#BLANK)
  (STATUS 37))

(DEFUN #END ()
  (#CAMERA NIL)
  (MSSG ((TIMES (STATUS 39) 9.999999E-4)) "SECONDS."))

; BLANK THE SCREENS.
; PUT A PROGRAM IN ADAGE TO PROVIDE A "FIXED" ADDRESS FOR THE
;                                     BEGINNING OF
; THE STACK DISPLAY, WHICH COMES DOWN FROM THE ADAGE LWA.
(DEFUN #BLANK ())

```

```

(OR STACK? (SETQ RMARGIN SCREEN-LIMIT))
(AND STAR (PUT STAR 'STAR NIL))
(AGTDSP 1 T FULLMODE CONTROL-LEN :CONTROL)
(AGTDSP LWA-ADAGE NIL FULLMODE 1 :EOL)
(SETQ TOP-SCREEN-AGT (SUB SCREEN-LIMIT TXTHT)
  STACK-TOP TOP-SCREEN-AGT
  BOTTOM-SCREEN-AGT BOTTOM-OF-FILM
  DISP-VARS NIL
  STAR NIL
  FOLLOW-X NIL
  FREE INIT-FREE
  FWA-STAR NIL
  FWA-MSGs NIL
  FWA-VARS NIL
  FWA-EVAL NIL
  LAST-STAR NIL)
(AND ?STEPMODE (#STEPMODE T)))

;PLOT THE SCREEN TO FILE -PLOT#
(DEFUN #PLOT ()
  (SETQ PLOTTED? T)
  (AGPLOT LWA-ADAGE))

;(#DEPTH N) SETS THE NUMBER OF LEVELS OF EVAL TO DISPLAY
(DEFUN #DEPTH (N)
  (SETQ #DEPTH N))

;START OR STOP CAMERA
(DEFUN #CAMERA (N)
  (SETQ CAMERA? (MEMQ N '(ON T)))
  (COND (CAMERA? (AGTCON LIGHTSON
    1
    0
    :CONTROL
    CAMERA-CONTROL)
    (OR TIME-FILM-STARTED
      (SETQ TIME-FILM-STARTED (STATUS 39))))
    (T (AGTCON LIGHTSOFF
      1
      0
      :CONTROL
      CAMERA-CONTROL)
      (AND TIME-FILM-STARTED
        (PROGN (SETQ FILM-USED (ADD FILM-USED
          (TIMES 9.999999E-4
            (SUB (STATUS
              39)
              TIME-FILM-STARTED))))
          (MSSG (FILM-USED)
            SECONDS
            OF
            FILM
            USED

```

```

                                ON
                                THIS
                                CARTRIDGE)
                                (SETQ TIME-FILM-STARTED NIL))))
(AGTDSP CAMERA-CONTROL
  NIL
  FULLMODE
  1
  :CONTROL
  CAMERA-CONTROL))

; (STEPMODE X) SETS STEP MODE. X MAY BE; ON, OFF, NIL, T
(DEFUN #STEPMODE N
  (SETQ RESTART-CAMERA (OR RESTART-CAMERA
    (AND CAMERA?
      (NOT ?STEPMODE)))
    ?STEPMODE (OR (EQ N 0) (MEMQ (ARG 1) '(ON T))))
  (COND (?STEPMODE (AGTDSP STEP-CONTROL
    NIL
    FULLMODE
    1
    :NOP)
    (AND CAMERA? (#CAMERA NIL)))
    (T (AGTDSP STEP-CONTROL
      NIL
      FULLMODE
      1
      :CONTROL
      STEP-CONTROL)
      (AND RESTART-CAMERA
        (#CAMERA T)
        (SETQ RESTART-CAMERA NIL))))))

; THIS BRINGS UP A PATTERN FOR ADJUSTING THE CAMERA
; LOW INTENSITY REDUCES SMEAR
(DEFUN #FOCUS ()
  (#BLANK)
  (#DISP (AGTCON INT 0 0.3 :ARRAY)
    (*GOTO NEG-SCREEN-LIMIT SCREEN-LIMIT)
    (*PLOT NEG-SCREEN-LIMIT BOTTOM-OF-FILM)
    (*PLOT SCREEN-LIMIT BOTTOM-OF-FILM)
    (*PLOT SCREEN-LIMIT SCREEN-LIMIT)
    (*PLOT NEG-SCREEN-LIMIT SCREEN-LIMIT)
    (*PLOT SCREEN-LIMIT BOTTOM-OF-FILM)
    (*GOTO NEG-SCREEN-LIMIT BOTTOM-OF-FILM)
    (*PLOT SCREEN-LIMIT SCREEN-LIMIT)
    (*GOTO 0 SCREEN-LIMIT)
    (*PLOT 0 BOTTOM-OF-FILM))
  (#WAIT 3600)
  (#BLANK))

; DISPLAY A GRID ON THE SCREEN FOR REFERENCE
(DEFUN #GRID ()
  (#DISP (DO Y

```



```

SCREEN-LIMIT
  (SUB Y 1.)
  (LESSP Y BOTTOM-OF-FILM)
  (AND (EQ Y 0.)
    (*INTENSITY BRIGHT)
    (AGTCVT 0 0 NIL NIL :ARRAY))
  (*GOTO NEG-SCREEN-LIMIT Y)
  (*PLOT SCREEN-LIMIT Y)
  (AND (EQ Y 0.) (*INTENSITY NORMAL)))
(DO X
  NEG-SCREEN-LIMIT
  (ADD X 1.)
  (GREATERP X SCREEN-LIMIT)
  (AND (EQ X 0.)
    (*INTENSITY BRIGHT)
    (AGTCVT 0 0 NIL NIL :ARRAY))
  (*GOTO X SCREEN-LIMIT)
  (*PLOT X BOTTOM-OF-FILM)
  (AND (EQ X 0.) (*INTENSITY NORMAL)))
(*DASH)
(*GOTO NEG-SCREEN-LIMIT BOTTOM-OF-FILM)
(*PLOT SCREEN-LIMIT BOTTOM-OF-FILM)
(*UNDASH)
(*SYMBOL 0.05 0.05 0 0.2))

```

```

;*****
; EVAL
;*****
(DEFUN #EVALQ NEXPR N
  (#EVAL (ARG 1)
    (COND ((EQ N 1) *ALIST)
      ((EVAL (ARG 2))))))

(DEFUN #EVAL N
  (AGTRD 0 EVAL-BUTTONS)
  (SETQ EVALING? T
    TEMP-FREE FREE
    DY-EVAL 0.
    DY-STACK 0.)
  (AND STACK?
    (SETQ RMARGIN (MIN RMARGIN
      (SUB STACK-X DONT-TOUCH))))
  (SETQ FWA-FOLLOW FREE
    FREE (ADD FWA-FOLLOW FOLLOW-LENGTH)
    FWA-RETVAL FREE
    FREE (ADD FWA-RETVAL AGT-VALUE-LENGTH))
  (AGTDSP (ADD1 FREE)
    NIL
    FULLMODE
    (SUB1 ROLL-EOL)
    :ROLL
    2)
  (AGTDSP FREE NIL FULLMODE 1 :ROLL)
  (AND FOLLOW?
    (AGTDSP (ADD1 FWA-FOLLOW)

```

```

      NIL
      FULLMODE
      (SUB1 FOLLOW-LENGTH)
      :FOLLOW
      2)
      (AGTDSP FWA-FOLLOW NIL FULLMODE 1 :FOLLOW))
(AND RETURNS?
  (AGTDSP FWA-RETVAL NIL FULLMODE 1 :SKIP-VAL))
(SETQ EVAL-ROLL FREE
  FWA-EVAL (ADD EVAL-ROLL ROLL-LEN)
  BOTTOM-EVAL (ADD EVAL-BLK-HT
    BOTTOM-OF-FILM
    DONT-TOUCH)
  NBLOCKS (ADD1 (FIX (DIVIDE (SUB TOP-SCREEN-AGT
    BOTTOM-EVAL)
    (ADD DOWN-Y
      EVAL-BLK-HT))))
  NROLL (FIX (ADD 0.5
    (TIMES NBLOCKS ROLL-RATIO)))
  ROLL-DIST (TIMES NROLL
    (ADD DOWN-Y EVAL-BLK-HT))
  JUMP-ROLL (ADD EVAL-ROLL ROLL-JUMP -1)
  STACK-BACKUP NIL
  EVAL-COUNTER (ADD 10000 EVAL-COUNTER)
  STACK-HEIGHT (SUB (SUB STACK-TOP 0.8)
    BOTTOM-OF-FILM)
  STACK-ADDRESS LWA-ADAGE
  Y-STACK BOTTOM-OF-FILM
  LAST-STAR NIL
  STACK-CHOP NIL
  LAST-RETVAL "" ")
(AND STACK?
  ALIST?
  (PROG (A LENGTH ADDR-LIST)
    (SETQ A (COND ((EQ N 1) *ALIST)
      ((ARG 2)))
    Y-STACK (ADD BOTTOM-OF-FILM
      (A-HEIGHT A))
    Y-ALIST (MAX BOTTOM-OF-FILM
      (SUB Y-STACK
        LINE-SKIP)))
    (AGTEXT (SUB STACK-X TXTHT)
      Y-ALIST
      TXTHT
      " ("
      :TEMP
      1)
    (INDEX :ARRAY 1)
    (MAPC 'ALIST-ENTRY A)
    (SETQ LENGTH (SUB1 (INDEX :ARRAY))
      STACK-ADDRESS (SUB LWA-ADAGE
        LENGTH))
    (MAPC '(LAMBDA (X)
      (ADDPROP (CAR X)
        'ALIST-ADDR
        (ADD STACK-ADDRESS
          (CDR X))))

```

```

        ADDR-LIST)
(AND (ZEROP LENGTH)
      (*SYMBOL STACK-X
        BOTTOM-OF-FILM
        ") "
        TXTHT)
      (SETQ Y-STACK (ADD Y-STACK
        LINE-SKIP)
        STACK-ADDRESS (SUB STACK-ADDRESS
        PAREN-LEN)
        LENGTH PAREN-LEN))
(AGTDSP STACK-ADDRESS
  NIL
  FULLMODE
  LENGTH
  :ARRAY)
(AGTCON JUMP STACK-ADDRESS 0 :ARRAY 1)
(AGTDSP STACK-JUMP NIL FULLMODE 1 :ARRAY)
(AGTDSP ALIST-LPAREN
  NIL
  FULLMODE
  PAREN-LEN
  :TEMP))
(INDEX :ARRAY 1)
(PROG1 (*EVAL (ARG 1)
  (COND ((EQ N 1) *ALIST)
    ((ARG 2)))
  NEG-SCREEN-LIMIT
  TOP-SCREEN-AGT
  0
  (LIST FWA-EVAL)
  (LIST STACK-ADDRESS)
  (LIST Y-STACK)
  T)
  (AGTRD 0 NORMAL-BUTTONS)
  (SETQ EVALING? NIL FREE TEMP-FREE)))

(SETQ *ALIST '((T . T)) *GOLIST NIL)

;FORM      TO BE EVALED
;A          THE ALIST
;X & Y     SCREEN COORDINATES
;LEVEL*    NUMBER OF LEVELS OF *EVAL
;LOCLIST   LIST OF ADAGE ADDRESSES OF BEGINNING OF EACH *EVAL
;          DISPLAY
;STACKLIST LIST OF ADDRESSES OF STACK DISPLAY
;STACK-Y   LIST OF Y-COORDINATES OF STACK TOP
;DISP?     NO DISPLAY IF NIL
(DEFUN *EVAL (FORM A X Y LEVEL* LOCLIST STACKLIST STACK-Y DISP?)
  (SETQ LEVEL* (ADD LEVEL* 1))
  (PROG (RETURN-VALUE BUFLOC OLD-DISP? MIDBOX RGT MID
    BOT PPSIZE EXIT-FORM STAR-DONE VAR1 VAR2
    #TIMES)
    (SETQ BUFLOC (CAR LOCLIST) OLD-DISP? DISP?)
    (AND (EQ LEVEL* (ADD1 #DEPTH))

```

```

      (SETQ DISP? NIL))
    (DO-SPECIALS)
    (AND DISP? (DO-EXCLUDES))
    (SETQ MID (ADD BOXW/2 X)
      DWN (SUB Y ARRL)
      PPSIZE 0
      BOT (SUB DWN BOXH)
      RGT (ADD X BOXW)
      MIDBOX (SUB DWN BOXH/2))
    (COND ((OR DISP? OLD-DISP?)
      (DISPEV-ENTER)
      (AND STACK-BACKUP (STACK-BACKUP))
      (#WAIT ENTER-WAIT))
      (T (SETQ LOCLIST (CONS BUFLOC LOCLIST))))
    (SETQ RETURN-VALUE (EVAL1 FORM))
    (COND ((OR DISP? OLD-DISP?)
      (DISPEV-EXIT)
      (#WAIT EXIT-WAIT)))
    (EVAL EXIT-FORM)
    (RETURN RETURN-VALUE))

```

```

(DEFUN EVAL1 (FORM)
  (COND ((NULL FORM) NIL)
    ((NUMBERP FORM) FORM)
    ((ATOM FORM)
      (*VALUE FORM
        A
        ' (ERRMSG UNDEFINED ATOM: (FORM))))
    ((ATOM (CAR FORM))
      (SELECTQ (CAR FORM)
        ' (CADR FORM)
        (COND (*EVCON (CDR FORM) A))
        (OR (*OR (CDR FORM)))
        (AND (*AND (CDR FORM)))
        (PROG (PROG1 (*PROG (CDR FORM) A)
          (UNBIND-DISP (CADR FORM))))
        (PROGN (*EVALIST (CDR FORM) A))
        (SETQ (FOLLOW (ADD X
          BOXW
          FOLLOWER-SIZE
          DONT-TOUCH)
          (SUB (ADD Y DY-EVAL)
            EVAL-BLK-HT)) (PROG1
            (*SETQ (CDR FORM)
              A)
              (FOLLOW-OFF)))
          (SET (FOLLOW (ADD X
            BOXW
            FOLLOWER-SIZE
            DONT-TOUCH)
            (SUB (ADD Y DY-EVAL)
              EVAL-BLK-HT)) (PROG1 (*SET
              (CDR FORM)

```

```

A)
(FOLLOW-OFF))
(ARG (*VALUE (MKATOM "ARG "
(*EVAL (CADR FORM)
A
(RIGHT)
(RIGHTY)
LEVEL*
LOCLIST
STACKLIST
STACK-Y
DISP?))
A
' (ERRMSG ARG
((CADR FORM))
IS
UNDEFINED)))
(RETURN (RETURN (*EVAL (CADR FORM)
A
(RIGHT)
(RIGHTY)
LEVEL*
LOCLIST
STACKLIST
STACK-Y
DISP?))
'*PROG))
(GO (SETQ PROGFORM (ASSOC (CADR FORM)
*GOLIST
' (ERRMSG NO
LABEL:
((CADR
FORM))
IN
PROG)))
" ")
(SELECT (*SELECT (SELECT-THING)
(CDDR FORM)))
(SELECTQ (*SELECTQ (SELECT-THING)
(CDDR FORM)))
(APPLY (APPLY '*APPLY
(NCONC (*EVLIS (CDDR FORM)
A)
A)))
(EVAL (*EVAL (PROG1 (*EVAL (CADR FORM)
A
(RIGHT)
(RIGHTY)
LEVEL*
LOCLIST
STACKLIST
STACK-Y
DISP?))
(WIPE-OUT))

```

A
X

```

(DOWN)
LEVEL*
LOCLIST
STACKLIST
STACK-Y
DISP?))

(PROG (EXP)
(COND ((SETQ EXP (OR (GET (CAR FORM)
                        'STAR)
                      (GET (CAR FORM)
                          'EXPR)))
      (*APPLY EXP
        (PROG1 (COND ((EQ (CAR
EXP)
                        'NLAMBDA)
                      (DISP-NF
(CDR FORM)))
                ((EQ (CAR
EXP)
                        'FLAMBDA)
                      (DISP-NF
(LIST (CDR FORM))))
                (T (*EVLIS
(CDR FORM)
A)))
      (AND (MEMQ (CAR
FORM)
BRIEF-FNS)
        (SETQ DISP?
NIL)))
      A))
((GET (CAR FORM) 'SUBR)
 (APPLY (CAR FORM)
        (*EVLIS (CDR FORM)
A)))
((GET (CAR FORM)
      'NSUBR)
 (ERRMSG NSUBR:
          ((CAR FORM)))
 (APPLY (CAR FORM)
        (CDR FORM)))
((GET (CAR FORM)
      'FSUBR)
 (ERRMSG FSUBR:
          ((CAR FORM)))
 (APPLY (CAR FORM)
        (LIST (CDR FORM))))
((CAR FORM)
 (*EVAL (CONS (*VALUE (CAR FORM)
A
' (ERRMSG
UNDEFINED

```



```

        ARGS))
        A))
    (UNBIND-DISP (CADR FN))))))
(T (*APPLY (*EVAL FN
    A
    (RIGHT)
    (RIGHTY)
    LEVEL*
    LOCLIST
    STACKLIST
    STACK-Y
    DISP?)
    ARGS
    A))))

```

```

(DEFUN *EVCON (C A)
  (COND ((NULL C) NIL)
        ((NULL (CDAR C))
         (COND ((NULL (CDR C))
                 (*EVAL (CAAR C)
                        A
                        X
                        (DOWN)
                        LEVEL*
                        LOCLIST
                        STACKLIST
                        STACK-Y
                        DISP?))
                ((*EVAL (CAAR C)
                        A
                        (RIGHT)
                        (RIGHTY)
                        LEVEL*
                        LOCLIST
                        STACKLIST
                        STACK-Y
                        DISP?))
                (T (WIPE-OUT) (*EVCON (CDR C) A))))))
  ((*EVAL (CAAR C)
          A
          (RIGHT)
          (RIGHTY)
          LEVEL*
          LOCLIST
          STACKLIST
          STACK-Y
          DISP?)
    (WIPE-OUT)
    (*EVALIST (CDAR C) A))
  (T (WIPE-OUT) (*EVCON (CDR C) A))))

```

```

(DEFUN *OR (L)
  (COND ((NULL L) NIL)
        ((*EVAL (CAR L)
                A

```



```

(RIGHT)
(RIGHTY)
LEVEL*
LOCLIST
STACKLIST
STACK-Y
DISP?))
(T (WIPE-OUT) (*OR (CDR L))))))

```

```

(DEFUN *AND (L)
  (COND ((NULL (CDR L))
    (*EVAL (CAR L)
      A
      (RIGHT)
      (RIGHTY)
      LEVEL*
      LOCLIST
      STACKLIST
      STACK-Y
      DISP?))
    ((*EVAL (CAR L)
      A
      (RIGHT)
      (RIGHTY)
      LEVEL*
      LOCLIST
      STACKLIST
      STACK-Y
      DISP?)
      (WIPE-OUT)
      (*AND (CDR L))))))

```

```

(DEFUN *EVLIS (M A)
  (COND ((AND DISP? EVLIS? EVLIS-MOVE?)
    (PROG (EV-LOC EVLIS-Y EV-RIGHT)
      (SETQ EVLIS-Y (SUB Y
        (ADD LINE-SKIP
          EVAL-BLK-HT))
        EV-LOC LOCLIST
        EV-RIGHT RIGHT-X)
      (PROG1 (MAPLIST '(LAMBDA (J)
        (DISP-EVLIS (*EVAL (CAR
          J)
          A
          (RIGHT EV-RIGHT)
          (RIGHTY)
          LEVEL*
          EV-LOC
          STACKLIST

```

```

STACK-Y
  DISP?)))
    M)
      (REALWAIT SHORT-WAIT)))
(T (MAPLIST ' (LAMBDA (J)
              (PROG1 (*EVAL (CAR J)
                      A
                      (RIGHT)
                      (RIGHTY)
                      LEVEL*
                      LOCLIST
                      STACKLIST
                      STACK-Y
                      (AND EVLIS?
                          DISP?))
                    (WIPE-OUT)))
    M))))

```

```

;RETURNS THE VALUE OF ATOM. IF NO VALUE
; IS FOUND, ERROR IS EVALED
(DEFUN *VALUE (ATOM ALIST ERROR)
  (PROG (VAL)
    (COND ((NULL ATOM) NIL)
          ((SETQ VAL (ASSOC ATOM ALIST))
            (AND REFS?
              (MEMQ ATOM DISP-VARS)
              (PROG (HIT)
                (FOLLOW (ADD X
                          BOXW
                          FOLLOWER-SIZE
                          DONT-TOUCH)
                  (SUB (ADD Y
                          DY-EVAL)
                      EVAL-BLK-HT))
                (FOLLOW X-FOLLOW-VARS
                  Y-FOLLOW-VARS)
                (FOLLOW-OFF)
                (AGTDSP (SETQ HIT (ADD FWA-VARS
                                      (GET ATOM
                                      'BEGIN)
                                      -1))
                  NIL
                  FULLMODE
                  1
                  :BRIGHT)
                (REALWAIT MEDIUM-WAIT)
                (AGTDSP HIT
                  NIL
                  FULLMODE
                  1
                  :NORMAL)
                (REALWAIT SHORT-WAIT)))
          (T (EVAL ERROR))))

```

;PAIR ATOMS WITH VALUES FOR ALIST. ALSO DOES NO-SPREAD BINDING

```
(DEFUN *PAIR (W Z)
  (COND ((LISTP W) (PAIR1 W Z))
        ((NULL W) NIL)
        (T (PROG (N)
                  (SETQ N 0)
                  (CONS (CONS W (LENGTH Z))
                        (MAPCAR '(LAMBDA (X)
                                   (CONS (MKATOM "ARG "
                                             (SETQ N
                                                (ADD1 N)))
                                         X))
                              Z))))))
```

```
(DEFUN PAIR1 (W Z)
  (COND ((NULL W) NIL)
        (T (CONS (CONS (CAR W) (CAR Z))
                  (PAIR1 (CDR W) (CDR Z))))))
```

;EVALS A LIST RETURNING THE LAST

```
(DEFUN *EVALIST (L A)
  (COND ((NULL (CDR L))
        (*EVAL (CAR L)
                A
                X
                (DOWN)
                LEVEL*
                LOCLIST
                STACKLIST
                STACK-Y
                DISP?))
        (T (*EVAL (CAR L)
                   A
                   X
                   (DOWN)
                   LEVEL*
                   LOCLIST
                   STACKLIST
                   STACK-Y
                   DISP?)
            (WIPE-OUT)
            (*EVALIST (CDR L) A))))
```

```
(DEFUN *PROG (D A)
  (PROG (*GOLIST)
        (SETQ *GOLIST (*GETLABELS (CDR D)))
        (*EVALPROG (CDR D)
                    (NCONC (STACK-DISP (*PAIRNIL (CAR D)))
                           A))))
```

;PAIRS A LIST OF PROG VARIABLES WITH NIL

```
(DEFUN *PAIRNIL (W)
```

```

(COND ((NULL W) NIL)
      (T (CONS (CONS (CAR W) NIL)
                (*PAIRNIL (CDR W))))))

;FIND LABELS IN A PROG. RETURNS LIST: ((LABEL . FORM) ( . . .))
(DEFUN *GETLABELS (W)
  (COND ((NULL W) NIL)
        ((LISTP (CAR W)) (*GETLABELS (CDR W)))
        (T (CONS (CONS (CAR W) (CDR W))
                  (*GETLABELS (CDR W))))))

;LIKE EVALIST, BUT IGNORES ATOMS. MTS STYLE PROG. EVAL CHANGES
;                                PROGFORM WHEN A "GO" IS DONE.
(DEFUN *EVALPROG (PROGFORM A)
  (COND ((NULL (CDR PROGFORM))
        (COND ((ATOM (CAR PROGFORM)) PROGFORM)
              (T (*EVAL (CAR PROGFORM)
                        A
                        X
                        (DOWN)
                        LEVEL*
                        LOCLIST
                        STACKLIST
                        STACK-Y
                        DISP?)
                  (COND ((CDR PROGFORM)
                        (*EVALPROG (CDR PROGFORM)
                                    A))))))
        (T (COND ((LISTP (CAR PROGFORM))
                  (*EVAL (CAR PROGFORM)
                        A
                        X
                        (DOWN)
                        LEVEL*
                        LOCLIST
                        STACKLIST
                        STACK-Y
                        DISP?)
                    (WIPE-OUT)))
            (*EVALPROG (CDR PROGFORM) A))))

(DEFUN *SETQ (Q A)
  (COND ((LISTP (CAR Q))
        (ERRMSG ATTEMPT TO SETQ: ((CAR Q)))
        ((NULL (CDDR Q))
         (*SETQ1 (CAR Q)
                  (*EVAL (CADR Q)
                        A
                        (RIGHT)
                        (RIGHTY)
                        LEVEL*
                        LOCLIST
                        STACKLIST
                        STACK-Y)
                    (CDDR Q))))))

```

```

                                DISP?)
                                A))
(T (*SETQ1 (CAR Q)
          (*EVAL (CADR Q)
                  A
                  (RIGHT)
                  (RIGHTY)
                  LEVEL*
                  LOCLIST
                  STACKLIST
                  STACK-Y
                  DISP?)

          A)
(WIPE-OUT)
(*SETQ (CDDR Q) A)))

(DEFUN *SET (Q A)
  (PROG (AT)
    (SETQ AT (*EVAL (CAR Q)
                    A
                    (RIGHT)
                    (RIGHTY)
                    LEVEL*
                    LOCLIST
                    STACKLIST
                    STACK-Y
                    DISP?))
    (WIPE-OUT)
    (COND ((LISTP (CAR Q))
            (ERRMSG ATTEMPT TO SET: (AT)))
           ((NULL (CDDR Q))
            (*SETQ1 AT
                    (*EVAL (CADR Q)
                            A
                            (RIGHT)
                            (RIGHTY)
                            LEVEL*
                            LOCLIST
                            STACKLIST
                            STACK-Y
                            DISP?)

                    A))
            (T (*SETQ1 AT
                    (*EVAL (CADR Q)
                            A
                            (RIGHT)
                            (RIGHTY)
                            LEVEL*
                            LOCLIST
                            STACKLIST
                            STACK-Y
                            DISP?)

                    A)
            (WIPE-OUT)
            (*SET (CDDR Q) A))))

```

```

;USED BY *SET *SETQ #SET #SETQ & CHANGE-VAR
(DEFUN *SETQ1 (W Z A)
  (COND ((EQ W (CAAR A))
    (RPLACD (CAR A) Z)
    (VAR-CHANGE W Z)
    (COND ((AND ALIST? (GET W 'ALIST-Y))
      (PROG (LENGTH ADDR Y)
        (INDEX :ARRAY 1)
        (PPD (CAR A)
          STACK-X
          (SETQ Y (GET W 'ALIST-Y))
          VALUE-LENGTH
          VAR-PPLEV
          0)
        (SETQ LENGTH (SUB1 (INDEX :ARRAY)))
        (FOLLOW (SUB STACK-X
          (ADD FOLLOWER-SIZE
            DONT-TOUCH))
          Y)
        (COND ((EQ Y BOTTOM-OF-FILM)
          (AND (LESSP LENGTH
            (SUB
              AGT-VALUE-LENGTH
              (PAREN-LEN))
            (AGTCON JUMPR
              (SUB
                AGT-VALUE-LENGTH
                (ADD LENGTH
                  PAREN-LEN))
                0
                :ARRAY))
          (AGTEXT "+X+"
            (ADD Y-ALIST
              LINE-SKIP)
            TXHT
            ") "
            :ARRAY
            (ADD1 (SUB
              AGT-VALUE-LENGTH
              (PAREN-LEN))))
          ((LESSP LENGTH
            AGT-VALUE-LENGTH)
            (AGTCON JUMPR
              (SUB AGT-VALUE-LENGTH
                LENGTH)
              0
              :ARRAY)))
        (AGTDSP (SETQ ADDR (SUB1 (GET W
          'ALIST-ADDR))))
        NIL
        FULLMODE
        1
        :SKIP-VAL)
        (AGTDSP (ADD1 ADDR)

```

```

NIL
FULLMODE
(SUB1 AGT-VALUE-LENGTH)
:ARRAY
2)
(AGTDSP ADDR
NIL
FULLMODE
1
:ARRAY)
(FOLLOW-OFF)
(REALWAIT SHORT-WAIT))))
Z)
((NULL (CDR A))
(NCONC A (CONS (CONS W Z) NIL))
(VAR-CHANGE W Z)
Z)
(T (*SETQ1 W Z (CDR A))))

```

```

(DEFUN SELECT-THING ()
  (SETQ EV-LOC LOCLIST)
  (COND (DISP? (PROG (EVLIS-Y)
    (SETQ EVLIS-Y (SUB Y
      (ADD LINE-SKIP
        EVAL-BLK-HT)))
    (PROG1 (DISP-EVLIS (*EVAL (CADR FORM)
      A
      (RIGHT)
      (RIGHTY)
      LEVEL*
      LOCLIST
      STACKLIST
      STACK-Y
      DISP?))
      (REALWAIT SHORT-WAIT))))))
  (T (*EVAL (CADR FORM)
    A
    (RIGHT)
    (RIGHTY)
    LEVEL*
    LOCLIST
    STACKLIST
    STACK-Y
    DISP?))))

```

```

(DEFUN *SELECT (THING CHOICES)
  (COND ((NULL (CDR CHOICES))
    (*EVAL (CAR CHOICES)
      A
      X
      (DOWN)
      LEVEL*
      EV-LOC
      STACKLIST
      STACK-Y

```

```

      DISP?))
    ((EQUAL THING
      (*EVAL (CAAR CHOICES)
        A
        (RIGHT)
        (RIGHTY)
        LEVEL*
        EV-LOC
        STACKLIST
        STACK-Y
        DISP?))
      (WIPE-OUT)
      (*EVALIST (CDAR CHOICES) A))
    (T (WIPE-OUT) (*SELECT THING (CDR CHOICES))))))

```

```

(DEFUN *SELECTQ (THING CHOICES)
  (COND ((NULL (CDR CHOICES))
    (*EVAL (CAR CHOICES)
      A
      X
      (DOWN)
      LEVEL*
      EV-LOC
      STACKLIST
      STACK-Y
      DISP?))
    ((COND ((ATOM (CAAR CHOICES))
      (EQ THING (CAAR CHOICES)))
      ((MEMQ THING (CAAR CHOICES))))
      (*EVALIST (CDAR CHOICES) A))
    ((*SELECTQ THING (CDR CHOICES)))))

```

```

;*****
;      DISPLAY
;*****
;DISPLAYS :ARRAY FOLLOWED BY EOL. RETURNS FWA
(DEFUN DISPL ()
  (PROG (N)
    (SETQ N (INDEX :ARRAY))
    (AGTCVT 0 0 NIL T :ARRAY)
    (AGTDSP FREE NIL FULLMODE 1 :EOL)
    (AGTDSP (ADD1 FREE)
      NIL
      FULLMODE
      (SUB1 N)
      :ARRAY
      2)
    (AGTDSP FREE NIL FULLMODE 1 :ARRAY)
    (PROG1 FREE (SETQ FREE (ADD FREE N -1)))))

```

```

;DISPLAYS :ARRAY FOLLOWED BY A JUMP TO THE STACK
(DEFUN DSP-EVAL (LOC LEN)
  (JUMP EVAL-RETURN)
  (AGTDSP (ADD1 LOC) NIL FULLMODE (SUB1 LEN) :ARRAY 2)

```



```

(AGTDSP LOC NIL FULLMODE 1 :ARRAY))

;ROLLS THE EVAL DISPLAY THE GIVEN DISTANCE (+=UP)
;ROLL-RATE IS DISPLAY-UNITS / SECOND
(DEFUN ROLL-EVAL (DIST)
  (PROG (SECS NFRAMES)
    (SETQ DY-EVAL (ADD DIST DY-EVAL)
      BOTTOM-EVAL (SUB BOTTOM-EVAL DIST)
      SECS (DIVIDE (ABS DIST) ROLL-RATE)
      DIST (TIMES DIST DY-DX-SCALE)
      NFRAMES (TIMES SECS FRAMES/SEC))
    (AGTCON INCR 0 (DIVIDE DIST NFRAMES) :TEMP 2)
    (SETQ NFRAMES (FIX (ABS (DIVIDE DIST
      (DIVIDE (CVT-AGTNUM
        (CADR (CONAGT :TEMP
          2)))
          8192.)))))
    (AGTCON SKIPZ NFRAMES 0 :TEMP 1)
    (AGTDSP EVAL-ROLL NIL FULLMODE 2 :TEMP)
    (AND RETURNS?
      (AGTDSP (ADD FWA-RETVAL 4)
        NIL
        FULLMODE
        2
        :TEMP))
    (REALWAIT (DIVIDE NFRAMES FRAMES/SEC))))

;CONVERT AGT NUMBER TO SIGNED INTEGER
(DEFUN CVT-AGTNUM (X)
  (COND ((GREATERP X 8192) (SUB X 16383))
    (X)))

;CHANGE A DISPLAYED ATOMIC VALUE
(DEFUN VAR-CHANGE (AT VAL)
  (COND ((AND (MEMQ AT DISP-VARS)
    (NEQ VAL (GET AT 'VALUE '*UNDEF* ")))
    (PROG (LENGTH ADDR)
      (AND (NUMBERP AT)
        (SETQ AT (MKATOM "ARG " AT)))
      (AND FOLLOW-X
        (FOLLOW X-FOLLOW-VARS
          Y-FOLLOW-VARS))
      (PUT AT 'VALUE VAL)
      (INDEX :ARRAY 1)
      (PPD VAL
        (GET AT 'X)
        (GET AT 'Y)
        VALUE-LENGTH
        VAR-PPLEV
        0)
      (SETQ LENGTH (SUB1 (INDEX :ARRAY)))
      (AND (LESSP LENGTH AGT-VALUE-LENGTH)
        (AGTCON JUMPR

```

```

        (SUB AGT-VALUE-LENGTH
          LENGTH)
      0
      :ARRAY))
    (SETQ ADDR (ADD FWA-VARS
                    (GET AT 'VAL-ADDR)
                    -1))
    (AGTDSP ADDR NIL FULLMODE 1 :SKIP-VAL)
    (AGTDSP (ADD1 ADDR)
            NIL
            FULLMODE
            (MIN LENGTH
              (SUB1 AGT-VALUE-LENGTH)))
      :ARRAY
      2)
    (AGTDSP ADDR NIL FULLMODE 1 :ARRAY)
    (REALWAIT MEDIUM-WAIT))))

```

```

;DISPLAY NEW STACK BLOCK
;THE NULL WORD IS TO STORE AN EOL WHEN ROLLED OFF SCREEN
(DEFUN STACK-DISP (L)
  (COND ((AND STACK? DISP?)
    (PROG (LENGTH ADDR ADDR-LIST)
      (SETQ Y-STACK (ADD DONT-TOUCH
                        (CAR STACK-Y)))
      (FOLLOW (ADD X
                  BOXW
                  FOLLOWER-SIZE
                  DONT-TOUCH)
        (SUB (ADD Y DY-EVAL)
              EVAL-BLK-HT))
      (FOLLOW (SUB STACK-X
                  (ADD FOLLOWER-SIZE
                      DONT-TOUCH))
        (ADD Y-STACK
              FOLLOWER-SIZE
              DY-STACK))
      (REALWAIT SHORT-WAIT)
      (AGTCVT 0 0 NIL NIL :ARRAY 1)
      (COND (ALIST? (SETQ Y-STACK (ADD (CAR STACK-Y)
                                        (A-HEIGHT L))
                                      Y-ALIST (SUB Y-STACK
                                                  LINE-SKIP)))
            (AGTEXT (SUB STACK-X
                          TXTHT)
                      Y-ALIST
                      TXTHT
                      "("
                      :TEMP
                      1)
            (AND L
              (MAPC 'ALIST-ENTRY
                    L)))
      (T (AND L (MAPC 'STACK-ENTRY L))
        (*DASH)
        (*GOTO STACK-X

```

```

      (SETQ Y-STACK (ADD DONT-TOUCH
                        Y-STACK)))
      (*PLOT SCREEN-LIMIT Y-STACK)
      (*UNDASH)))
    (SETQ LENGTH (SUB1 (INDEX :ARRAY))
      STACKLIST (CONS (SETQ ADDR (SUB (CAR
      STACKLIST)
      LENGTH)))
      STACKLIST)
      STACK-Y (CONS Y-STACK STACK-Y))
    (AND (GREATERP Y-STACK STACK-TOP)
      (STACK-ROLL))
    (AGTDSP ADDR
      NIL
      FULLMODE
      LENGTH
      :ARRAY)
    (AGTCON JUMP ADDR 0 :ARRAY 1)
    (AND ALIST?
      (AGTDSP ALIST-LPAREN
        NIL
        FULLMODE
        PAREN-LEN
        :TEMP)
      (MAPC ' (LAMBDA (X)
        (ADDPROP (CAR X)
          'ALIST-ADDR
          (ADD ADDR
            (CDR X))))
        ADDR-LIST))
    (AGTDSP STACK-JUMP
      NIL
      FULLMODE
      1
      :ARRAY)
    (AND L
      (MAPC ' (LAMBDA (X)
        (VAR-CHANGE (CAR X)
          (CDR X)))
        L))
    (FOLLOW-OFF)))
  (T (COND ((AND DISP? L)
    (FOLLOW (ADD X
      BOXW
      FOLLOWER-SIZE
      DONT-TOUCH)
      (SUB (ADD Y DY-EVAL)
        EVAL-BLK-HT))
      (MAPC ' (LAMBDA (X)
        (VAR-CHANGE (CAR X)
          (CDR X)))
        L)
      (FOLLOW-OFF)))
    (SETQ STACKLIST (CONS (CAR STACKLIST)
      STACKLIST)
      STACK-Y (CONS (CAR STACK-Y) STACK-Y))))

```

L)

```

;PUT A VALUE ON THE STACK
(DEFUN STACK-ENTRY (AT)
  (SETQ AT (CAR AT))
  (COND ((OR (EQ STACK-VARS T) (MEMQ AT STACK-VARS))
    (AGTEXT STACK-X
      Y-STACK
      TXTHT
      (MKATOM AT "" =")
      :ARRAY)
    (PPD (*VALUE AT A ""*UNDEF* ")
      (ADD STACK-X
        (TIMES TXTHT (ADD 3 (PLEN AT))))
      Y-STACK
      VALUE-LENGTH
      VAR-PPLEV
      0)
    (SETQ Y-STACK (ADD Y-STACK LINE-SKIP))))))

```

```

;MAKE A-LIST ENTRY ON STACK DISPLAY
(DEFUN ALIST-ENTRY (L)
  (COND ((OR (EQ STACK-VARS T)
    (MEMQ (CAR L) STACK-VARS))
    (SETQ ADDR (INDEX :ARRAY))
    (PPD L
      STACK-X
      Y-ALIST
      VALUE-LENGTH
      VAR-PPLEV
      0)
    (SETQ ADDR-LIST (CONS (CONS (CAR L) ADDR)
      ADDR-LIST))
    (ADDPROP (CAR L) 'ALIST-Y Y-ALIST)
    (SETQ LENGTH (SUB (INDEX :ARRAY) ADDR)
      Y-ALIST (SUB Y-ALIST LINE-SKIP))
    (COND ((LESSP Y-ALIST BOTTOM-OF-FILM)
      (AND (LESSP LENGTH
        (SUB AGT-VALUE-LENGTH
          PAREN-LEN))
        (AGTCON JUMPR
          (SUB AGT-VALUE-LENGTH
            (ADD LENGTH
              PAREN-LEN))
          0
          :ARRAY))
      (AGTEXT "+X+"
        (ADD Y-ALIST LINE-SKIP)
        TXTHT
        ") "
        :ARRAY
        (ADD ADDR
          (SUB AGT-VALUE-LENGTH
            PAREN-LEN))))
    ((LESSP LENGTH AGT-VALUE-LENGTH)
      (AGTCON JUMPR

```

```

(SUB AGT-VALUE-LENGTH
  LENGTH)
0
:ARRAY)))
(INDEX :ARRAY (ADD ADDR AGT-VALUE-LENGTH))))))

```

```

; DETERMINE THE HEIGHT OF AN ALIST DISPLAY ENTRY
(DEFUN A-HEIGHT (L)
  (PROG (N)
    (AND (EQ STACK-VARS T)
      (RETURN (TIMES LINE-SKIP (LENGTH L))))
    (SETQ N 0)
    (MAPC ' (LAMBDA (X)
      (AND (MEMQ (CAR X) STACK-VARS)
        (SETQ N (ADD N LINE-SKIP))))
      L)
    (RETURN N)))

```

```

; DECIDES HOW MUCH TO ROLL THE STACK
(DEFUN STACK-ROLL ()
  (PROG (N LIST OLD)
    (SETQ LIST STACK-Y N 1 OLD STACK-CHOP)
    LOOP
      (AND (MINUSP (ADD (CAR LIST) DY-STACK))
        (GO DONE))
      (SETQ N (ADD1 N) LIST (CDR LIST))
      (GO LOOP)
    DONE
    (AGTDSP (SETQ STACK-CHOP (CAR (NTH STACKLIST
      N))))
    NIL
    FULLMODE
    1
    :EOL)
    (AND OLD (AGTDSP OLD NIL FULLMODE 1 :NOP))
    (ROLL-STACK (SUB BOTTOM-OF-FILM
      (ADD DY-STACK (CAR LIST))))))

```

```

; ROLL THE STACK DISPLAY
(DEFUN ROLL-STACK (DIST)
  (PROG (SECS NFRAMES)
    (SETQ DY-STACK (ADD DY-STACK DIST)
      SECS (DIVIDE (ABS DIST) ROLL-RATE)
      STACK-TOP (SUB STACK-TOP DIST)
      DIST (TIMES DIST DY-DX-SCALE)
      NFRAMES (TIMES SECS FRAMES/SEC))
    (COND ((NEQ 0 NFRAMES)
      (AGTCON INCR
        0
        (DIVIDE DIST NFRAMES)
        :TEMP
        2)
      (SETQ NFRAMES (FIX (ABS (DIVIDE DIST
        (DIVIDE

```

```
(CVT-AGTNUM (CADR (CONAGT :TEMP
```

```
2)))
```

```
8192.)))))
```

```
(AGTCON SKIPZ NFRAMES 0 :TEMP 1)
```

```
(AGTDSP STACK-ROLL-ADDR
```

```
NIL
```

```
FULLMODE
```

```
2
```

```
:TEMP)
```

```
(REALWAIT (DIVIDE NFRAMES FRAMES/SEC))))))
```

```
;CHANGE DISPLAY AS STACK POPS
```

```
(DEFUN UNBIND-DISP (LIST)
```

```
(AND LIST
```

```
(ATOM LIST)
```

```
(PROG (N L)
```

```
(SETQ N 0 L NIL)
```

```
(REPEAT '(SETQ L (CONS (MKATOM "ARG "
```

```
(SETQ N (ADD1 N)))
```

```
L))
```

```
(LENGTH ARGS))
```

```
(SETQ LIST (CONS LIST L)))
```

```
(COND ((AND STACK? DISP?)
```

```
(FOLLOW-OFF)
```

```
(FOLLOW (ADD X
```

```
BOXW
```

```
FOLLOWER-SIZE
```

```
DONT-TOUCH)
```

```
(SUB (ADD Y DY-EVAL) EVAL-BLK-HT))
```

```
(FOLLOW (SUB STACK-X
```

```
(ADD FOLLOWER-SIZE DONT-TOUCH))
```

```
(ADD (CAR STACK-Y)
```

```
FOLLOWER-SIZE
```

```
DY-STACK))
```

```
(COND (ALIST? (MAPC '(LAMBDA (AT)
```

```
(REM AT
```

```
'ALIST-ADDR)
```

```
(REM AT
```

```
'ALIST-Y))
```

```
LIST)
```

```
(AGTEXT (SUB STACK-X TXTHT)
```

```
(SUB (CADR STACK-Y)
```

```
LINE-SKIP)
```

```
TXTHT
```

```
"("
```

```
:TEMP
```

```
1)
```

```
(AGTDSP ALIST-LPAREN
```

```
NIL
```

```
FULLMODE
```

```
PAREN-LEN
```

```
:TEMP)))
```

```
(AGTCON JUMP (CADR STACKLIST) 0 :TEMP 1)
```

```
(AGTDSP STACK-JUMP NIL FULLMODE 1 :TEMP)
```

```

(SETQ STACK-Y (CDR STACK-Y)
  STACKLIST (CDR STACKLIST))
(ROLL-STACK-BACK)
(MAPC '(LAMBDA (AT)
  (VAR-CHANGE AT
    (*VALUE AT
      A
      '*UNDEF* ")))
  LIST)
(FOLLOW-OFF))
((AND LIST DISP?)
  (FOLLOW-OFF)
  (FOLLOW (ADD X
    BOXW
    FOLLOWER-SIZE
    DONT-TOUCH)
    (SUB (ADD Y DY-EVAL) EVAL-BLK-HT))
  (MAPC '(LAMBDA (AT)
    (VAR-CHANGE AT
      (*VALUE AT
        A
        '*UNDEF* ")))
    LIST)
  (FOLLOW-OFF)))

```

```

(DEFUN ROLL-STACK-BACK ()
  (COND ((AND (LESSP (ADD DY-STACK DONT-TOUCH) 0.)
    (LESSP (CAR STACK-Y)
      (SUB STACK-TOP STACK-HEIGHT)))
    (ROLL-STACK (MIN STACK-HEIGHT
      (MINUS DY-STACK)))
    (PROG (N LIST OLD)
      (SETQ LIST STACK-Y N 1 OLD STACK-CHOP)
      LOOP
        (OR (GREATERP (ADD (CAR LIST)
          DY-STACK)
          BOTTOM-OF-FILM)
          (GO DONE))
        (SETQ N (ADD1 N) LIST (CDR LIST))
        (GO LOOP)
      DONE
      (AGTDSP (SETQ STACK-CHOP (CAR (NTH STACKLIST
        N)))
        NIL
        FULLMODE
        1
        :EOL)
      (AND OLD
        (AGTDSP OLD NIL FULLMODE 1 :NOP))))))

```

```

;RESTORE THE STACK IF WE BACK UP
(DEFUN STACK-BACKUP ()
  (SETQ STACK-BACKUP NIL)
  (AGTCON JUMP (CAR STACKLIST) 0 :TEMP 1)
  (AGTDSP STACK-JUMP NIL FULLMODE 1 :TEMP)

```

```
(ROLL-STACK-BACK)
(DISP-ALL-VARS))
```

```
;MAKE SURE ALL DISPLAYED VARIABLES ARE CORRECT
(DEFUN DISP-ALL-VARS ()
  (MAPC '(LAMBDA (AT)
    (VAR-CHANGE AT
      (*VALUE AT A '"*UNDEF* ")))
    DISP-VARS))
```

```
;THIS CLEARS THE RETURN VALUE
(DEFUN WIPE-VAL ()
  (COND ((AND RETURNS? (NEQ LAST-RETVAL '" ")))
    (AGTDSP FWA-RETVAL NIL FULLMODE 1 :SKIP-VAL)
    (SETQ LAST-RETVAL '" "))))
```

```
(DEFUN WIPE-OUT ()
  (AND DISP? (REALWAIT SHORT-WAIT) (WIPE-VAL)))
```

```
;CALCULATE POSITION OF EVAL BLOCK TO THE RIGHT, AND DRAW AN
; ARROW
```

```
(DEFUN RIGHT N
  (INDEX :ARRAY 1)
  (COND (DISP? (*GOTO RGT MIDBOX)
    (*PLOT (SETQ X-ARROW (ADD X
      (COND ((ZEROP N)
        RIGHT-X)
      ((MAX (ARG 1)
        RIGHT-X))))))
    (SETQ Y-ARROW (SUB Y
      (ADD PPSIZE
        DONT-TOUCH))))
    (SETQ SLOPE (DIVIDE (SUB MIDBOX
      Y-ARROW)
      (SUB RGT X-ARROW)))
    (*PLOT (SUB X-ARROW
      (TIMES (ADD 1. SLOPE)
        BARBSIZE))
      (ADD Y-ARROW
        (TIMES (SUB 1.
          (TIMES SLOPE
            0.5))
          BARBSIZE)))
    (*GOTO (SUB X-ARROW
      (TIMES (SUB 1.
        (TIMES SLOPE
          0.5))
        BARBSIZE))
      (SUB Y-ARROW
        (TIMES (ADD 1. SLOPE)
          BARBSIZE)))
    (*PLOT X-ARROW Y-ARROW).
```



```

      (ADD X-ARROW DONT-TOUCH))
(X))

```

```

;CALCULATE THE "Y" POSITION OF AN EVAL BLOCK PLACED TO THE
;
      RIGHT

```

```

(DEFUN RIGHTY ()
  (COND (DISP? (SUB Y (ADD PPSIZE TXTHT DONT-TOUCH)))
    (Y)))

```

```

;CALCULATE POSITION OF EVAL BLOCK DOWNWARDS, AND DRAW AN ARROW
(DEFUN DOWN ()

```

```

  (INDEX :ARRAY 1)
  (COND (DISP? (*GOTO MID BOT)
    (*PLOT MID
      (SETQ Y-ARROW (MIN (SUB Y
        (ADD PPSIZE
          TXTHT
          DONT-TOUCH))
        (ADD TXTHT
          (SUB Y
            (ADD DOWN-Y

```

```

          EVAL-BLK-HT))))))
    (*PLOT (ADD MID BARBSIZE)
      (ADD Y-ARROW BARBSIZE))
    (*GOTO (SUB MID BARBSIZE)
      (ADD Y-ARROW BARBSIZE))
    (*PLOT MID Y-ARROW)
    (SUB Y-ARROW TXTHT))
(Y))

```

```

;CALLED WHEN *EVAL IS ENTERED
;IF DISP? WAS JUST CHANGED TO NIL, MAKE THREE DOTS
;IF WE JUST STARTED DISPLAYING AFTER OMITTING SOME, UPDATE ALL
;
      DISPLAYED VALUES

```

```

(DEFUN DISPEV-ENTER ()
  (COND (OLD-DISP?)
    (T (DISP-ALL-VARS) (SETQ Y (DOWN))))
  (COND (DISP? (COND ((LESSP Y BOTTOM-EVAL)
    (AGTCON JUMP
      (CAR (NTH LOCLIST
        (ADD1 (SUB NBLOCKS
          NROLL))))
      0
      :TEMP
      1)
    (AGTDSP JUMP-ROLL
      NIL
      FULLMODE
      1
      :TEMP)
    (ROLL-EVAL ROLL-DIST)))
    (SETQ PPSIZE (TIMES LINE-SKIP
      (PPD FORM

```

```

(MAX X
  (SUB (ADD X
        BOXW/2)
    (TIMES TXTHT/2
      (PLEN
        (COND ((ATOM FORM)
          FORM)
        ((CAR FORM))))))
  Y
  (FIX (DIVIDE (SUB
    RMARGIN
      X)
    TXTHT))
    EVAL-PPLEV
    LINE-SKIP))
  LEN1 (INDEX :ARRAY))
  (AGTDSP BUFLOC NIL FULLMODE 1 :STACK)
  (DSP-EVAL BUFLOC LEN1)
  (REALWAIT SHORT-WAIT)
  (INDEX :ARRAY 1)
  (*GOTO MID (SUB Y ITSY-BITSY))
  (*PLOT MID DWN)
  (*PLOT (ADD MID BARBSIZE)
    (ADD DWN BARBSIZE))
  (*GOTO (SUB MID BARBSIZE)
    (ADD DWN BARBSIZE))
  (*PLOT MID DWN)
  (*PLOT RGT DWN)
  (*PLOT RGT BOT)
  (*PLOT X BOT)
  (*PLOT X DWN)
  (*PLOT MID DWN)
  (*SYMBOL (ADD X EV-HT)
    (ADD BOT EV-*)
    'EVAL
    EV-HT)
  (SETQ LEN2 (INDEX :ARRAY))
  (DSP-EVAL (ADD BUFLOC LEN1 -1) LEN2)
  (SETQ LOCLIST (CONS (ADD BUFLOC
    LEN2
    LEN1
    -2)
    LOCLIST)))
(T (*GOTO X Y)
  (*PLOT X Y)
  (*GOTO (ADD X BOXW/2) Y)
  (*PLOT (ADD X BOXW/2) Y)
  (*GOTO (ADD X BOXW) Y)
  (*PLOT (ADD X BOXW) Y)
  (SETQ LEN1 (INDEX :ARRAY)
    PPSIZE 0
    MIDBOX (SUB (SUB Y ARRL) BOXH/2))
  (DSP-EVAL BUFLOC LEN1)
  (SETQ LOCLIST (CONS (ADD BUFLOC LEN1 -1)
    LOCLIST))))

```

```

;CALLED WHEN EVAL EXITS
;BLANKS OUT EVAL BLOCK, MOVES OR DISPLAYS RETURN VALUE, AND
;
;      ROLLS IF NECESSARY
;IF WE WERE NOT PREVIOUSLY DISPLAYING, UPDATE ALL VARIABLES
(DEFUN DISPEV-EXIT ()
  (AGTDSP BUFLOC NIL FULLMODE 1 :STACK)
  (AND STAR-DONE
    LAST-STAR
    (AGTDSP LAST-STAR NIL FULLMODE 1 :NORMAL)
    (SETQ LAST-STAR NIL))
  (COND (RETURNS? (PROG (L)
    (COND ((EQ RETURN-VALUE
      LAST-RETVAL)
      (MOVE-VAL X Y)
      (SETQ LAST-RETVAL-X X
        LAST-RETVAL-Y Y))
      (T (WIPE-VAL)
        (AGTCON INT
          0
          BRIGHT
          :ARRAY
          1)
        (AGTCON SKIPZ
          0
          0
          :ARRAY)
        (AGTCON INCR
          0
          0
          :ARRAY)
        (AGTCON DX
          0
          0
          :ARRAY)
        (AGTCON SKIPZ
          0
          0
          :ARRAY)
        (AGTCON INCR
          0
          0
          :ARRAY)
        (AGTCON DY
          0
          (TIMES DY-EVAL
            DY-DX-SCALE)
          :ARRAY)
        (AGTCVT 0
          0
          NIL
          NIL
          :ARRAY)
        (PPD RETURN-VALUE
          (SETQ LAST-RETVAL-X X)
          (SETQ LAST-RETVAL-Y

```

```

MIDBOX)
    VALUE-LENGTH
    VAR-PPLEV
    0)
    (SETQ L (SUB1 (INDEX :ARRAY))
      LAST-RETVAL RETURN-VALUE)
    (AND (LESSP L
      AGT-VALUE-LENGTH)
      (AGTCON JUMPR
        (SUB
          AGT-VALUE-LENGTH
            L)
          0
          :ARRAY))
    (AGTDSP (ADD1 FWA-RETVAL)
      NIL
      FULLMODE
      (SUB1 (MIN (ADD1 L)
        AGT-VALUE-LENGTH))
      :ARRAY
      2)
    (AGTDSP FWA-RETVAL
      NIL
      FULLMODE
      1
      :ARRAY)
    (OR DISP?
      (DISP-ALL-VARS))))))
(COND ((AND (GREATERP DY-EVAL DONT-TOUCH)
  (GREATERP Y (ADD BOTTOM-EVAL ROLL-DIST)))
  (ROLL-EVAL (MINUS (MIN DY-EVAL ROLL-DIST)))
  (AGTCON JUMP
    (CAR (COND ((EQ DY-EVAL 0.)
      (LAST LOCLIST))
    ((NTH LOCLIST
      (MIN (LENGTH LOCLIST)
        (ADD1 NBLOCKS))))))
    0
    :ARRAY
    1)
  (AGTDSP JUMP-ROLL NIL FULLMODE 1 :ARRAY))))

; MOVE THE RETURN VALUE TO X Y.
(DEFUN MOVE-VAL (X Y)
  (PROG (SECS NFRAMES DIST)
    (SETQ DIST (MAX (ABS (SUB X LAST-RETVAL-X))
      (ABS (SUB Y LAST-RETVAL-Y)))
    SECS (DIVIDE DIST MOVE-RATE)
    NFRAMES (FIX (TIMES SECS FRAMES/SEC)))
  (COND ((NEQ NFRAMES 0)
    (AGTCON SKIPZ NFRAMES 0 :TEMP 1)
    (AGTCON INCR
      0
      (TIMES DY-DX-SCALE
        (DIVIDE (SUB X

```

```

                                LAST-RETVAL-X)
                                NFRAMES))
                                :TEMP)
(AGTDSP (ADD1 FWA-RETVAL)
        NIL
        FULLMODE
        2
        :TEMP)
(AGTCON INCR
        0
        (TIMES DY-DX-SCALE
          (DIVIDE (SUB Y
                      LAST-RETVAL-Y)
                  NFRAMES)))
        :TEMP
        2)
(AGTDSP (ADD 4 FWA-RETVAL)
        NIL
        FULLMODE
        2
        :TEMP)
(REALWAIT SECS))))

```

```

;DISPLAY A LIST OF EVLIS VALUES

```

```

(DEFUN DISP-EVLIS (VAL)
  (PROG (L)
    (COND ((EQ VAL LAST-RETVAL)
           (MOVE-VAL X EVLIS-Y)))
    (INDEX :ARRAY 1)
    (PPD VAL X EVLIS-Y EVLIS-PPLEN VAR-PPLEV 0)
    (SETQ EV-RIGHT (SUB "+X+" X)
           L (INDEX :ARRAY
                    EVLIS-Y (SUB EVLIS-Y LINE-SKIP)))
    (JUMP EVAL-RETURN)
    (AGTDSP (ADD1 (CAR EV-LOC))
            NIL
            FULLMODE
            (SUB1 L)
            :ARRAY
            2)
    (WIPE-VAL)
    (AGTDSP (CAR EV-LOC) NIL FULLMODE 1 :ARRAY)
    (SETQ EV-LOC (CONS (ADD (CAR EV-LOC) L -1)
                      (CDR EV-LOC))))
  VAL)

```

```

;DISPLAY N OR FLAMBDA EVLIS

```

```

(DEFUN DISP-NF (VAL)
  (AND DISP?
        EVLIS?
        EVLIS-MOVE?
        (PROG (EVLIS-Y)
          (SETQ EVLIS-Y (SUB Y
                              (ADD LINE-SKIP
                                  EVAL-BLK-HT)))

```

```

      EV-LOC LOCLIST)
      (DISP-EVLIS VAL)
      (REALWAIT SHORT-WAIT)))
VAL)

```

```

;TAKE CARE OF SPECIAL FORMS INTERSPERSED WITH FUNCTION
;      DEFINITIONS
;      (# (ENTRY-FORM) (FORM) (EXIT-FORM))
;      THIS IS A USER INSERTION. ENTRY-FORM IS EVALED (BY
;      MTS LISP, NOT
;      BY THE ANIMATION LISP) BEFORE FORM IS EVALED FOR
;      ANIMATION.
;      EXIT-FORM IS EVALED UPON EXIT.
;      BOTH FORMS ARE INVISIBLE TO THE ANIMATION.
;      (* ADDR (FORM))
;      ADDR IS ADAGE LOCATION TO BE FILLED WITH AN
;      INTENSIFY INSTRUCTION
(DEFUN DO-SPECIALS ()
  (COND ((AND (ATOM (CAR FORM))
              (EQNAME (CAR FORM) USER-FORM))
        (SETQ #TIMES (COND ((EQ (CAR FORM)
                                USER-FORM)
                          (RPLACA FORM
                                (COPY USER-FORM))
                          (SET (CAR FORM) (ADD1
                                EVAL-COUNTER))
                          1)
              ((LESSP (CAAR FORM)
                      EVAL-COUNTER)
               (SET (CAR FORM) (ADD1
                                EVAL-COUNTER))
              1)
              (T (SET (CAR FORM) (ADD1 (CAAR
                                FORM))))
              (SUB (CAAR FORM)
                    EVAL-COUNTER))))))
    (PROG (RET)
      (SETQ RET (EVAL (CADR FORM)))
      (OR (CDDR FORM) (RETURN RET '*EVAL)))
    (SETQ EXIT-FORM (CADDR FORM)
          FORM (CADDR FORM))
    (DO-SPECIALS))
  ((EQ (CAR FORM) STAR-FORM)
   (DO-STAR (CADR FORM))
   (SETQ FORM (CADDR FORM))
   (DO-SPECIALS))))

(DEFUN DO-STAR (ADDR)
  (AND LAST-STAR
       (AGTDSP LAST-STAR NIL FULLMODE 1 :NORMAL))
  (COND ((AND DISP? ADDR)
         (AGTDSP (SETQ STAR-DONE T
                       LAST-STAR (ADD FWA-STAR ADDR))
                 NIL
                 FULLMODE

```

```

      1
      :BRIGHT))
(T (SETQ LAST-STAR NIL)))

```

```

;PROCESS EXCLUDED ATOMS AND FUNCTIONS
(DEFUN DO-EXCLUDES ()
  (COND ((AND (ATOM FORM) (MEMQ FORM EXCLUDE-ATOMS))
    (RETURN (*VALUE FORM
      A
      ' (ERRMSG UNDEFINED
        ATOM:
        (FORM)))
      '*EVAL))
    ((AND (ATOM (CAR FORM))
      (MEMQ (CAR FORM) EXCLUDE-FNS))
      (SETQ DISP? NIL OLD-DISP? NIL))))

```

```

;THIS ALLOWS FUNCTIONS WITH BREAKPOINTS TO BE EVALUED BY MTS
;
;LISP FOR TESTING
(DEFUN # FEXPR (L)
  (OR (CDR L)
    (PRINT "WARNING-BREAKPOINT WITH ENTRY-FORM ONLY"))
  (EVAL (CADR L)))

```

```

;THIS IS USED TO DISPLAY THINGS IN THE EXIT FORM
(DEFUN #EXIT-DISP FEXPR (L)
  (INDEX :ARRAY 1)
  (EVLIS (CDR L))
  (DSP-EVAL BUFLOC (INDEX :ARRAY))
  (REALWAIT (EVAL (CAR L))))

```

```

;MOVE FOLLOWER TO X, Y
;DISPLAY AT X, Y IF NOT ON ALREADY
;IT DOES NOT APPEAR UNTIL IT STARTS TO MOVE
(DEFUN FOLLOW (X Y)
  (AND DISP?
    FOLLOW?
    (COND (FOLLOW-X (PROG (SECS NFRAMES DIST)
      (SETQ DIST (MAX (ABS (SUB X
        FOLLOW-X))
          (ABS (SUB Y
            FOLLOW-Y))))
        SECS (DIVIDE DIST
          FOLLOW-RATE)
        NFRAMES (FIX (TIMES SECS
          FRAMES/SEC))))
      (COND ((NEQ NFRAMES 0)
        (AGTCON SKIPZ
          NFRAMES
          0

```

```

:TEMP
1)
      (AGTCON INCR
0
      (TIMES DY-DX-SCALE
      (DIVIDE (SUB
X
FOLLOW-X)
NFRAMES))
:TEMP)
      (AGTDSP (ADD1 FWA-FOLLOW)
NIL
FULLMODE
2
:TEMP)
      (AGTCON INCR
0
      (TIMES DY-DX-SCALE
      (DIVIDE (SUB
Y
FOLLOW-Y)
NFRAMES))
:TEMP
2)
      (AGTDSP (ADD 4
FWA-FOLLOW)
NIL
FULLMODE
2
:TEMP)
      (AGTDSP FWA-FOLLOW
NIL
FULLMODE
1
:BRIGHT)
      (REALWAIT SECS))))
(T (AGTCON DY
0
      (TIMES Y DY-DX-SCALE)
:TEMP
1)
      (AGTDSP (ADD FWA-FOLLOW 6)
NIL
FULLMODE
1
:TEMP)
      (AGTCON DX
0
      (TIMES X DY-DX-SCALE)
:TEMP
1)
      (AGTDSP (ADD FWA-FOLLOW 3)
NIL

```



```

                                FULLMODE
                                1
                                :TEMP)))
(SETQ FOLLOW-X X FOLLOW-Y Y)))

;TURN OFF FOLLOWER
(DEFUN FOLLOW-OFF ()
  (AND FOLLOW?
    FOLLOW-X
    (AGTDSP FWA-FOLLOW NIL FULLMODE 1 :FOLLOW)
    (SETQ FOLLOW-X NIL)))

;*****
;      SHOW
;*****
;(#SHOW S <X <Y>>) DISPLAY CONS CELLS OF S
(DEFUN #SHOW FEXPR (F)
  (PROG (X Y)
    (SETQ X (OR (EVAL (CADR F)) NEG-SCREEN-LIMIT)
          Y (OR (EVAL (CADDR F))
                (SUB SCREEN-LIMIT ATMHT)))
    (INDEX :ARRAY 1)
    (SHW1 (EVAL (CAR F)) X Y)
    (DISPL)))

;(#SHOWFN FN <X <Y>>) SHOW FUNCTION FN
(DEFUN #SHOWFN FEXPR (L)
  (INDEX :ARRAY 1)
  (*INTENSITY BRIGHT)
  (AGTCVT 0 0 NIL NIL :ARRAY)
  (PROG (X Y)
    (SETQ X (OR (EVAL (CADR L)) NEG-SCREEN-LIMIT)
          Y (OR (EVAL (CADDR L))
                (SUB SCREEN-LIMIT TXTHT*2)))
    (*SYMBOL X
             Y
             (MKATOM (FNAME (CAR L)) ':)
             TXTHT)
    (*INTENSITY NORMAL)
    (SHW1 (GET (CAR L) 'EXPR) X (SUB Y TXTHT*2))
    (DISPL)))

(DEFUN SHW1 (F X Y)
  (PROG (FROMLIST-CAR TOLIST-CAR FROMLIST-CDR
        TOLIST-CDR DONELIST CAR-COUNT)
    (SETQ CAR-COUNT 0)
    (AND LOOPS? (LISTP F) (SCAN-LOOPS F))
    (SHW F X Y)
    (DRAW-LOOPS FROMLIST-CAR TOLIST-CAR LOOPDEC)
    (DRAW-LOOPS FROMLIST-CDR
                 TOLIST-CDR
                 (ADD LOOPDEC SOMESPACE))))

```

```

;LOOK FOR LOOPS IN SHOW DISPLAY
(DEFUN SCAN-LOOPS (F)
  (AND (ATOM (CAR F))
    (EQNAME (CAR F) USER-FORM)
    (SETQ F (CADDR F)))
  (SETQ DONELIST (CONS F DONELIST))
  (COND ((ATOM (CAR F))
    ((MEMQ (CAR F) DONELIST)
      (SETQ FROMLIST-CAR (CONS F FROMLIST-CAR))
      (SETQ TOLIST-CAR (CONS (LIST (CAR F))
        TOLIST-CAR)))
    (T (SCAN-LOOPS (CAR F)))))
  (COND ((ATOM (CDR F))
    ((MEMQ (CDR F) DONELIST)
      (SETQ FROMLIST-CDR (CONS F FROMLIST-CDR))
      (SETQ TOLIST-CDR (CONS (LIST (CDR F))
        TOLIST-CDR)))
    (T (SCAN-LOOPS (CDR F)))))

;SAVE THE "TO" COORDINATES IN TOLIST
(DEFUN DO-TOLIST (ENT)
  (AND (EQ F (CAR ENT))
    (RPLACD ENT (LIST X (ADD Y CONSIZE/2)))))

;DRAW ANY LOOPS IN THE SHOW DISPLAY
(DEFUN DRAW-LOOPS (FROM TO LOOPDEC)
  (COND ((NULL FROM)
    (T (PROG (A Y Y2 X2)
      (SETQ X (CADAR FROM) Y (CADDR FROM))
      (*GOTO X Y)
      (*PLOT X (SETQ Y2 (ADD Y LOOPDEC)))
      (SETQ X (CADAR TO))
      (COND ((LESSP (CADDR TO) Y)
        (SETQ X2 (SUB X CONSIZE)
          Y (ADD (CADDR TO)
            CONSIZE/2)))
        ((SETQ X2 (SUB X
          (SUB CONSIZE
            SOMESPACE))
          Y (SUB (CADDR TO)
            CONSIZE/2)))))
      (*PLOT X2 Y2)
      (*PLOT X2 Y)
      (*PLOT X Y)
      (*PLOT (SUB X SHOW-BARBSIZE)
        (SUB Y SHOW-BARBSIZE))
      (*GOTO (SUB X SHOW-BARBSIZE)
        (ADD Y SHOW-BARBSIZE))
      (*PLOT X Y)
      (DRAW-LOOPS (CDR FROM)
        (CDR TO)
        LOOPDEC)))))

```



```

                                                    X)
                                                    Y))
      ' (0. 0.))
      (T (SHW (CDR F)
            XD
            Y)))
(OR TEMP2
  (RROW (COND ((EQ (CAR SIZED) 0)
                (ADD X
                     MINCONSWIDTH))
          (XD))))
(LIST (MAX (CAR SIZEA)
          (SUB (ADD XD (CAR SIZED))
               X))
      (MAX (ADD YDEC (CADR SIZEA))
            (CADR SIZED))))))

;DISPLAY AN ATOM
(DEFUN DISPATM (F Y)
  (COND (DETAILATOM? (CONSBX)
                    (*SYMBOL (ADD X
                                   CONSIZE*2
                                   SOMESPACE)
                              Y
                              F
                              ATMHT)
          (COND ((EQ F (CAR F))
                  (*GOTO DWN# Y)
                  (*PLOT DWN#
                        (SUB Y CONSIZE))
                  (COND ((EQ F (CDR F))
                          (*GOTO (ADD DWN#
                                       CONSIZE)
                                Y)
                          (*PLOT (ADD DWN#
                                   CONSIZE)
                                (SUB Y
                                   CONSIZE))
                          (*PLOT DWN#
                                (SUB Y
                                   CONSIZE))
                          (*PLOT (ADD DWN#
                                       SHOW-BARBSIZE)
                                (ADD (SUB Y
                                       CONSIZE)
                                   SHOW-BARBSIZE))
                          (*GOTO (ADD DWN#
                                       SHOW-BARBSIZE)
                                (SUB (SUB Y
                                       CONSIZE)
                                   SHOW-BARBSIZE))
                          (*PLOT DWN#
                                (SUB Y
                                   CONSIZE))))
                  (*PLOT (SUB DWN# CONSIZE)
                          (SUB Y CONSIZE))
                  )
          )

```

```

(*PLOT (SUB DWN# CONSIZE)
      Y)
(*PLOT X Y)
(*PLOT (SUB X
      SHOW-BARBSIZE)
      (SUB Y
      SHOW-BARBSIZE))
(*GOTO (SUB X
      SHOW-BARBSIZE)
      (ADD Y
      SHOW-BARBSIZE))
      (*PLOT X Y)))
      (LIST (ADD CONSIZE*2
      (TIMES (PLEN F) ATMHT))
      ATMHT))
(T (*SYMBOL (ADD X SOMESPACE) Y F ATMHT)
  (LIST (MAX MINCONSWIDTH
      (TIMES ATMHT (PLEN F)))
      ATMHT)))

```

```

;DISPLAY A CONS CELL

```

```

(DEFUN DISPCONS ()
  (CONSBOX)
  (*GOTO DWN# Y)
  (*PLOT DWN# BOT#)
  (*PLOT (SUB DWN# SHOW-BARBSIZE)
      (ADD BOT# SHOW-BARBSIZE))
  (*GOTO (ADD DWN# SHOW-BARBSIZE)
      (ADD BOT# SHOW-BARBSIZE))
  (*PLOT DWN# BOT#))

```

```

(DEFUN CONSBOX ()
  (SETQ TOP# (ADD Y CONSIZE)
      MID# (ADD X CONSIZE)
      RIGHT# (ADD X CONSIZE*2)
      DWN# (ADD X CONSIZE/2)
      BOT# (ADD Y YDEC CONSIZE))
  (*GOTO X Y)
  (*PLOT X TOP#)
  (*PLOT RIGHT# TOP#)
  (*PLOT RIGHT# Y)
  (*PLOT X Y)
  (*GOTO MID# TOP#)
  (*PLOT MID# Y))

```

```

;DRAW AN ARROW TO RIGHT AND SLASH CONS BOXES IF CDR=NIL

```

```

(DEFUN RARROW (RIGHT)
  (COND ((OR SHOWNIL? (CDR F))
      (SETQ MID (ADD Y CONSIZE/2))
      (*GOTO (ADD X CONSIZE*2) MID)
      (*PLOT RIGHT MID)
      (*PLOT (SUB RIGHT SHOW-BARBSIZE)
      (ADD MID SHOW-BARBSIZE))
      (*GOTO (SUB RIGHT SHOW-BARBSIZE)

```

```

                (SUB MID SHOW-BARBSIZE))
        (*PLOT RIGHT MID))
    (T (*GOTO (ADD X CONSIZE) Y)
      (*PLOT (ADD X CONSIZE*2) (ADD Y CONSIZE))))))

;IF SIZEA=NIL WE RAN OFF BOTTOM (CAR) OTHERWISE OFF RIGHT
(DEFUN OFFSCREEN ()
  (COND (CONTINUE? (PPD F
                        X
                        Y
                        (FIX (DIVIDE (SUB SCREEN-LIMIT
                                      X)
                                      ATMHT))
                        5
                        LINE-SKIP)))
    '(0. 0.))

```

```

;*****
;      PRETTYPRINTER
;*****
; PPD RETURNS THE # OF LINES DISPLAYED
;IF LINE-SKIP IS 0, ONLY ONE LINE WILL BE PRINTED
(DEFUN PPD (FORM PX PY MAXWIDTH LEVEL LINE-SKIP)
  (PROG (*PLEN TEST-FLAG "+Y+" "+SIZE+" Y-DEC
        SINGLE-FLAG)
    (SETQ *PLEN 0 "+X+" PX "+Y+" PY)
    (SETQ Y-DEC LINE-SKIP
      "+SIZE+" (TIMES TXTHT 0.857))
    (*PPFORM1 FORM LEVEL)
    (*TERPRI*)
    (ABS (DIVIDE (SUB "+Y+" PY)
      (COND ((EQ Y-DEC 0) 1)
      (Y-DEC))))))

```

```

(DEFUN *PPFORM1 (FORM LEV)
  (SETQ LEV (SUB1 LEV))
  (COND ((ATOM FORM) (WR FORM))
    ((LESSP LEV 1) (WR '&))
    ((AND SHORTQUOTE (EQ (CAR FORM) 'QUOTE))
      (WR '"'))
    (*PPFORM1 (CADR FORM) LEV))
  ((AND (ATOM (CAR FORM))
    (EQNAME (CAR FORM) USER-FORM))
    (AND (CDDR FORM)
      (*PPFORM1 (CADDR FORM) LEV)))
  ((EQ (CAR FORM) STAR-FORM)
    (SET-STAR FORM '(*PPFORM1 (CADDR FORM) LEV)))
  ((AND (OR (LESSP LEV 2)
    (NOT (MEMQ (CAR FORM) '(COND DEFUN))))
    (*PPFIT FORM)))
  ((*PPSPECIAL FORM))
  ((PROG (CARFORM TAB)
    (SETQ CARFORM (UNCONS FORM FORM))
    (WR "(")

```

```

(COND ((ATOM CARFORM)
      (WR (FNAME CARFORM))
      (*SKIP* 1)
      (SETQ TAB (ADD1 *PLEN))
      (COND ((LISTP FORM)
              (*PPFORM1 (UNCONS FORM
                           FORM)
                           LEV))
              (FORM (WR ".")
                     (*SKIP* 1)
                     (WR FORM)
                     (SETQ FORM NIL))
              (T (*SKIP* -1))))
      (T (SETQ TAB (PLUS 2 *PLEN))
          (*PPFORM1 CARFORM LEV)))
(MAPC ' (LAMBDA (CARFORM)
          (*TERPRI*)
          (*TAB* TAB)
          (*PPFORM1 CARFORM LEV))
      FORM)
(AND (CDR (LAST FORM))
     (*SKIP* 1)
     (WR ".")
     (*SKIP* 1)
     (WR (CDR (LAST FORM))))
(WR ")")

```

```

(DEFUN *PPFIT (FORM)
  (COND ((GREATERP MAXWIDTH
                    (PLUS *PLEN
                          (TESTLEN FORM PX PY 0 T)))
        (*PRIN1* FORM LEV)
        T)))

```

```

(DEFUN *PPSPECIAL (FORM)
  (COND ((AND (ATOM (CAR FORM))
              (NOT (NUMBERP (CAR FORM))))
        (APPLY1 (GET (CAR FORM)
                      '*PPSPECIAL
                      '(RETURN NIL '*PPSPECIAL))
                 FORM)
        T)))

```

```

(DEFUN *PPPLOG (FORM)
  (WR "(")
  (WR (UNCONS FORM FORM))
  (*SKIP* 1)
  (PROG (TAB VARS)
        (SETQ TAB (ADD1 *PLEN))
        (COND ((*PPFIT (CAR FORM)))
              (T (SETQ VARS (CAR FORM))
                  (WR "(")
                  (*PRIN1* (CAR VARS) LEV)
                  (MAPC ' (LAMBDA (CARFORM)

```

```

(*SKIP* 1)
(COND ((*PPFIT CARFORM))
  (T (*TERPRI*)
    (*TAB* TAB)
    (*PRIN1* CARFORM
      LEV))))
      (CDR VARS))
    (WR " ")))
  (MAPC ' (LAMBDA (FORM)
    (*TERPRI*)
    (*TAB* TAB)
    (COND ((ATOM FORM)
      (*SKIP* -2)
      (WR FORM))
    ((*PPFORM1 FORM LEV))))
    (CDR FORM))
  (WR " ")))

```

```

(DEFUN *PPSETQ (FORM)
  (WR " (")
  (WR (CAR FORM))
  (*SKIP* 1)
  (PROG (TAB SW)
    (SETQ TAB (ADD1 *PLEN) SW T FORM (CDR FORM))
    (REPEAT ' (PROGN (OR SW (*TERPRI*) (*TAB* TAB))
      (*PPFORM1 (CAR FORM) LEV)
      (*SKIP* 1)
      (*PPFORM1 (CADR FORM) LEV)
      (SETQ FORM (CDDR FORM) SW NIL))
    (SHIFT (LENGTH FORM) -1)))
  (WR " )"))

```

```

(DEFUN *PPDEFUN (FORM)
  (WR " (")
  (WR (UNCONS FORM FORM))
  (*SKIP* 1)
  (WR (FNAME (UNCONS FORM FORM)))
  (*SKIP* 1)
  (COND ((MEMQ (CAR FORM) ' (EXPR FEXPR NEXPR))
    (WR (UNCONS FORM FORM))
    (*SKIP* 1)))
  (PROG (VARS)
    (SETQ VARS (UNCONS FORM FORM))
    (COND (VARS (*PPFORM1 VARS LEV))
      ((WR " ( "))))
  (MAPC ' (LAMBDA (CARFORM)
    (*TERPRI*)
    (*TAB* 3)
    (*PPFORM1 CARFORM LEV))
    FORM)
  (WR " )"))

```

```

(DEFPROP DEFUN *PPSPECIAL *PPDEFUN)

```



```
(DEFPROP PROG *PPSPECIAL *PPPROG)
```

```
(DEFPROP (SETQ SET) *PPSPECIAL *PPSETQ)
```

```
(DEFUN *TAB* (N)
  (OR SINGLE-FLAG
    (SETQ "+X+" (ADD PX (TIMES (SUB1 N) "+SIZE+"))
      *PLEN (MAX (SUB1 N) *PLEN)))))
```

```
(DEFUN *TERPRI* ()
  (COND ((EQ 0 LINE-SKIP) (SETQ SINGLE-FLAG T))
    (T (SETQ *PLEN 0
      "+X+" PX
      "+Y+" (SUB "+Y+" Y-DEC)))))
NIL)
```

```
(DEFUN *SKIP* (N)
  (AND (GREATERP N 0) (SETQ BLANK "+X+"))
  (SETQ "+X+" (ADD "+X+" (TIMES N "+SIZE+"))
    *PLEN (ADD *PLEN N)))
```

```
(DEFUN TESTLEN (F +X+ +Y+ *PLEN TEST-FLAG)
  (*PRIN1* F LEV)
  *PLEN)
```

```
(DEFUN *PRIN1* (F LEV)
  (COND ((ATOM F) (WR F))
    ((AND (ATOM (CAR F))
      (EQNAME (CAR F) USER-FORM))
      (AND (CDDR F) (*PRIN1* (CADDR F) LEV)))
    ((EQ (CAR F) STAR-FORM)
      (SET-STAR F '(*PRIN1* (CADDR F) LEV)))
    ((AND SHORTQUOTE (EQ (CAR F) 'QUOTE))
      (WR "'")
      (*PRIN1* (CADR F) (SUB1 LEV)))
    ((LESSP LEV 1) (WR '&))
    ((AND (ATOM (CAR F)) (GET (CAR F) 'ALIAS))
      (WR "(")
      (WR (FNAME (CAR F)))
      (AND (CDR F) (*SKIP* 1) (*PRIN2 (CDR F)))
      (WR ")"))
    (T {WR "(" (*PRIN2 F) (WR ")"})))
```

```
(DEFUN *PRIN2 (F)
  (COND ((NULL F))
    ((NULL (CDR F)) (*PRIN1* (CAR F) (SUB1 LEV)))
    ((ATOM (CDR F))
      (*PRIN1* (CAR F) (SUB1 LEV))
      (*SKIP* 1))
```

```

(WR ".")
(*SKIP* 1)
(*PRIN1* (CDR F) (SUB1 LEV)))
(T (*PRIN1* (CAR F) (SUB1 LEV))
  (*SKIP* 1)
  (*PRIN2 (CDR F)))))

```

;WRITE AN ATOM ON THE SCREEN

```

(DEFUN WR (F)
  (COND ((NOT TEST-FLAG)
    (AND SINGLE-FLAG (SETQ F "" ...))
    (AND (NUMBERP F) (SETQ F (MKATOM F)))
    (AGTEXT "+X+" "+Y+" "+SIZE+" F :ARRAY)
    (AND SINGLE-FLAG (RETURN 1 'PPD))))
  (SETQ *PLEN (ADD *PLEN (PLEN F))
    "+X+" (ADD "+X+" (TIMES "+SIZE+" (PLEN F)))))

```

;PLACE ADAGE ADDR IN STAR-FORM AND INSERT INTENSIFY INSTRUCTION

```

(DEFUN SET-STAR (FRM EXP)
  (COND ((AND (NOT TEST-FLAG) STAR-FLAG)
    (SETQ LAST-INTS T)
    (RPLACA (CDR FRM) (SUB1 (INDEX :ARRAY)))
    (*INTENSITY NORMAL)
    (EVAL EXP)
    (COND (LAST-INTS (*INTENSITY NORMAL)
      (SETQ LAST-INTS NIL))))
    (T (EVAL EXP))))

```

```

;*****
; GRAPHICS PRIMITIVES
;*****

```

```

(DEFUN #DISP FEXPR (L)
  (INDEX :ARRAY 1)
  (EVLIS L)
  (DISPL))

```

```

(DEFUN *PLOT (X Y)
  (AGTCVT X Y T NIL :ARRAY))

```

```

(DEFUN *JUMP NEXPR (LABEL)
  (PROG (L)
    (SETQ L (COND ((ASSQ LABEL LABELS))
      (T (SETQ LABELS (CONS (LIST LABEL
        NIL)
        LABELS))
      (CAR LABELS)))))
    (NCONC L
      (LIST (CONS JUMPR
        (INDEX :ARRAY
          (ADD1 (INDEX :ARRAY))))))))

```

```
(DEFUN JUMP (N)
  (AGTCON JUMP N 0 :ARRAY))
```

```
(DEFUN *GOTO (X Y)
  (AGTCVT X Y NIL NIL :ARRAY))
```

```
(DEFUN *SKIPNZ (COUNT)
  (AGTCON SKIPNZ COUNT 0 :ARRAY))
```

```
(DEFUN *SKIPZ (COUNT)
  (AGTCON SKIPZ COUNT 0 :ARRAY))
```

```
(DEFUN *SYMBOL (X Y STRING SIZE)
  (AND (NUMBERP STRING) (SETQ STRING (MKATOM STRING)))
  (AGTEXT X Y SIZE STRING :ARRAY))
```

```
(DEFUN *INTENSITY (V)
  (AGTCON INT 0 V :ARRAY))
```

```
(DEFUN *DASH ()
  (AGTCON DASH 0 0 :ARRAY))
```

```
(DEFUN *UNDASH ()
  (AGTCON UNDASH 0 0 :ARRAY))
```

```
(DEFUN FOR FEXPR (L)
  (GPROG (*SKIPNZ (FIX (TIMES FRAMES/SEC
                        (EVAL (CAR L))))))
  (*JUMP OUT)
  (EVLIS (CDR L))
  OUT))
```

```
;THIS ACCEPTS A GLISP SUBROUTINE AND DISPLAYS IT IMMEDIATELY
(DEFUN SUBROUTINE FEXPR (L)
  (INDEX :ARRAY 1)
  (AGTCON SCALE 0 GLISP-SCALE :ARRAY)
  (SETQ SUBROUTINE? T)
  (EVLIS (CDR L))
  (SETQ SUBROUTINE? NIL)
  (AGTCON SCALE 0 *SCALE :ARRAY)
  (DISPL))
```

```
;THIS IS SIMILAR TO GLISP ADJ EXCEPT THAT DIALS ARE NOT ALLOWED
(DEFUN ADJ FEXPR (L)
  (OR FADE?
    (AGTCON INT
      0
```

```

      (TIMES (EVAL (CADDAR L)) 0.1)
      :ARRAY))
(AGTCON DX 0 (TIMES (EVAL (CAAR L)) 0.1) :ARRAY)
(AGTCON DY 0 (TIMES (EVAL (CADAR L)) 0.1) :ARRAY)
(AGTCON SCALE
  0
  (TIMES (EVAL (CAR (CDDDAR L))) 0.1)
  :ARRAY)
(EVLIS (CDR L))
(OR FADE? (AGTCON INT 0 NORMAL :ARRAY))
(AGTCON DX 0 0 :ARRAY)
(AGTCON DY 0 0 :ARRAY)
(AGTCON SCALE
  0
  (COND (SUBROUTINE? GLISP-SCALE)
    (*SCALE))
  :ARRAY))

;DRAW THE VECTORS GENERATED BY PRIMITIVES ONE AT A TIME
(DEFUN #DRAW FEXPR (L)
  (AGTCON SCALE 0 GLISP-SCALE :ARRAY 1)
  (EVLIS (CDR L))
  (AGTCON SCALE 0 *SCALE :ARRAY)
  (AGTCVT 0 0 NIL T :ARRAY)
  (PROG (Z N LIM)
    (SETQ Z 1
      LIM (INDEX :ARRAY)
      N (FIX (EVAL (CAR L))))
    (COND ((GREATERP N 0)
      (PROG NIL
        LOOP
          (AGTDSP FREE
            NIL
            FULLMODE
            1
            :ARRAY
            Z)
          (RTWAIT N)
          (SETQ FREE (ADD1 FREE)
            Z (ADD1 Z))
          (AND (LESSP Z LIM) (GO LOOP))))
      (T (PROG NIL
        (SETQ N (MAX 1 (ABS N)))
        LOOP
          (AGTDSP FREE
            NIL
            FULLMODE
            N
            :ARRAY
            Z)
          (SETQ FREE (ADD N FREE)
            Z (ADD N Z))
          (AND (LESSP Z LIM) (GO LOOP)))))))

(DEFUN GPROG FEXPR (GPROG)

```

```

(PROG (LABELS)
  (MAPC ' (LAMBDA (GPROG)
    (COND ((ATOM GPROG)
      (LABL GPROG))
      ((EVAL GPROG))))
    GPROG)
  (RESOLVELABELS)))

```

```

(DEFUN RESOLVELABELS ()
  (PROG (LOC SAVEINDEX REFS)
    (SETQ SAVEINDEX (INDEX :ARRAY))
    (MAPC ' (LAMBDA (L)
      (COND ((SETQ LOC (CADR L))
        (MAPC ' (LAMBDA (REF)
          (AGTCON (CAR REF)
            (SUB LOC
              (CDR
                REF)))
            0
            :ARRAY
            (CDR
              (CDDR L))))
          (T (ERRMSG "UNKNOWN LABEL"
            ((CAR L)))
            (RETURN NIL))))
        LABELS)
    (INDEX :ARRAY SAVEINDEX)))

```

```

(DEFUN LABL (LABEL)
  (PROG (L)
    (SETQ L (COND ((ASSQ LABEL LABELS))
      (T (SETQ LABELS (CONS (LIST LABEL
        NIL)
          LABELS)))
        (CAR LABELS))))
    (AND (CADR L)
      (PROGN (ERRMSG (LABEL)
        "IS MULTIPLY DEFINED")
        (RETURN NIL)))
    (RPLACA (CDR L) (INDEX :ARRAY)))

```

```

(SETQ LABELS NIL)

```

```

; (FADE HOLDTIME FADETIME DISPLAY-CODE)
(DEFUN FADE FEXPR (L)
  (SETQ FADEFRAMES (FIX (TIMES FRAMES/SEC
    (EVAL (CADR L))))
    HOLDFRAMES (FIX (TIMES FRAMES/SEC
      (EVAL (CAR L))))
  (GPROG (*SKIPNZ (ADD FADEFRAMES HOLDFRAMES))
    (*JUMP OUT)
    (*SKIPNZ HOLDFRAMES))

```

```

(AGTCON INCR
  0
  (MINUS (DIVIDE NORMAL FADEFRAMES))
  :ARRAY)
(*INTENSITY NORMAL)
(SETQ FADE? T)
(EVLIS (CDDR L))
(SETQ FADE? NIL)
(*INTENSITY NORMAL)
OUT))

```

```

(DEFUN ERRMSG FEXPR #L#
  (TERPRI)
  (PRIN1 '***ERROR:))
  (MESSAGE)
  (TERPRI)
  (AND EVALING? (PRIN1 'FORM:) (PRIN1 FORM) (TERPRI)))

```

```

(DEFUN MSSG FEXPR #L#
  (TERPRI)
  (MESSAGE)
  (TERPRI))

```

```

(DEFUN MESSAGE ()
  (MAPC ' (LAMBDA (#X#)
    (COND ((EQ #X# ':) (TERPRI) (TAB 5))
      ((ATOM #X#) (PRIN1 #X#))
      ((MAPC ' (LAMBDA (#X#)
        (PRIN1 (EVAL #X#)))
        #X#))))))
  (ARG 1)))

```

```

(AND ATAGT? (#START))

```

```

PRINT OFF
$CONTINUE WITH LISP:LISPSTART RETURN
PRINT ON
*LISP MACROS WERE READ IN ABOVE BUT NOT PRINTED
*
*      IFADG      RETURN T IF AT THE ADAGE, NIL OTHERWISE
*
IFADG      CSECT
           BALR 8,0
           USING *,8
           STM 13,6,SAVE
           CALL GUINFO,(C74,RNAM)
           LM 13,6,SAVE
           LR PDS,D
           LR A,NILR
           L 9,RNAM      GET THE DEVICE NAME
           C 9,ADAGE     SEE IF DS27
           BNE OUT       RETURN NIL IF NOT
           L A,TRUEATOM  RETURN T--DEFINED IN LISP MACROS
OUT        BR RET
ADAGE      DC C'DS27'
RNAM       DS 6F        ROOM FOR NAME AND USER COMMENT
C74        DC F'74'     CODE FOR DEVICE NAME
SAVE       DS 10F       REGISTER SAVE AREA
END

```