LOGO: AN APPROACH TO COMPUTER-BASED LEARNING

by

KIMBERLY ANN ARMSTRONG POLLACK

B. A., University of California, Berkeley, 1975

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

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming to the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

May, 1979

Department of Computer Science

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

Date May 3, 1979

ABSTRACT

The objective of this thesis is to explore the possible use of LOGO as a vehicle for computer assisted instruction in both secondary and post-secondary environments. The spectrum of current languages and systems for CAI is summarized by way of selected example languages and some of the most recent LOGO-based research is described. In particular, the work performed at BBN, MIT and at the University of Edinburgh is summarized. The remainder of the thesis is based on the author's experience in using LOGO at the University of British Columbia. Included are a new LOGO User's Manual and the results of an evaluation of both the language and the Manual as well as a discussion of the actions taken based on the students' responses. The thesis concludes with a comparison of LOGO with its nearest competitor (BASIC), a critique of the BCLOGO implementation and a final chapter containing the author's conclusions and proposals for future research.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I would like to thank my supervisor, Dr. Alan Mackworth, for his many ideas, support, patience and careful editing of this thesis. I would also like to thank Dr. Abbe Mowshowitz and Vincent Manis for their comments and my husband, Bary, for his encouragement and understanding.

LOGO: AN APPROACH TO COMPUTER-BASED LEARNING

1. Introduction

Although the first uses of computing were largely for military and scientific computations, the major use today is for data processing and administration. This usage is characterized by large volumes of input and output and relatively simple computations. Data processing entails, among other things, creating, updating and deleting records, recording transactions, sorting and printing statements [BOHL, 1971]. These processes arise in the normal operation of business, institutions and government. Other uses of computers include generation of music, playing chess, checkers, or bridge, generating pictures for animated films and controlling machinery.

Why should computers in schools be confined to tasks such as "compute the sum of the squares of the first twenty odd numbers"? Why not use computers to help children learn fundamental skills which are needed in everyday life; let us use the computers to help educate the next generation. This thesis describes one approach to using computers in the educational process.

Unfortunately, when educators speak about computers in education they do not all have the same image in mind. Some think of using the computer to program the child; others think of teaching the child to program the computer. But most educators have at least this belief in common: the transactions between the computer and the child will be some kind of conversation or questions and answers using words or numbers.

Computers may be used to help children with various levels of ability and intelligence. They may be used to help those students who are bright and find the everyday routine of the classroom boring. These students can use computers to help them in advanced subjects while the teacher is busy with other students. The computer may be used by those students who are slow or who are having difficulty in a particular subject. The computer can give these students the individual help which they need. Interaction with computers can be an enjoyable learning experience for all students no matter how quickly or slowly they learn if a proper environment is provided.

This thesis is about LOGO, a high-level advanced computer language designed especially for the beginner. It was developed jointly by Bolt, Beranek and Newman, Inc., and Seymour Papert, a consultant from the Artificial Intelligence Laboratory at MIT. Both BBN and Papert have been using LOGO to help children learn mathematical skills, to aid children who have mental disabilities and to teach children to learn how to think.

One of the features which makes LOGO especially attractive for instructing beginners is its facilities for "turtles". A turtle is a disk about one foot in diameter equipped with wheels, a light, a horn and a pen found on the bottom in the center of the disk. The turtle is run by commands given to it by the child with the aid of a button box. The buttons specify the following actions: Forward, Backward, Right, Left, Go, Stop, Pendown, Penup and Honk. When the pen is down the turtle traces the path it is taking.

Several studies have been done on children who are low in scholastic skills; typically, after only one month of individual work with the computer these skills have improved and the children have again become interested in learning.

The objectives of this thesis are to present a description of LOGO, to evaluate the claims of its proponents, to critically review the LOGO approach, to expose the reader to a few CAI languages with a description of each and to evaluate students' reactions to LOGO. In Chapter 2 I discuss the area of computers in education -- how computers are used for instruction in public schools throughout British Columbia and how computers have helped simplify administrative tasks. I also discuss Computer Assisted Instruction and present a summary of the SOPHIE, PLATO, EUCLID, PILOT and NATAL-74 systems. Chapter 3 discusses the history of LOGO and summarize what BBN, MIT and Edinburgh are doing with the language. Additionally, there is a section describing the other languages with which LOGO has to compete. Chapter 4 presents an overall summary of the LOGO language. LOGO is compared with LISP and there is a short section on control and data structures. Chapter 5 describes the materials I have developed for making use of LOGO. Chapter 6 presents a summary of my experience in CPSC 200, a course in which LOGO and my manual were used, also the results of the questionnaire, listing each statement with students' responses. Chapter 7 presents a critique of LOGO. And in Chapter 8 I present my conclusions and some thoughts for future work.

2. Computers in Education

2.1 Survey within British Columbia

The use of computers in education is becoming very common: out of approximately 90 school districts throughout British Columbia 25 (including all Lower Mainland schools) have computer access of one form or another [DODDS, 1978]. Available hardware ranges from time-shared medium- to large-scale computer systems which are owned or rented by the schools to portable micro-computers which are also owned or rented by the schools. Although only BASIC is being taught and used on most of these systems many students are sponsored to go to UBC or SFU and learn additional languages -- PASCAL, APL BCPL and others.

At the eleventh and twelfth grade levels full semester computer courses are available; in grade levels eight, nine and ten there are units on computers that the students may elect to take. In all levels students write programs and do simulations in such areas as business applications, economics and mathematics. Many students become very involved with computers and spend most of their spare time learning more about their organization, structure and use, as well as learning additional languages.

2.2 Administrative Use

The administrative use of computers in the secondary and elementary schools may be divided into two general categories: business applications and teacher related tasks.

At the Ministerial, School District and School levels, the education system is a "business", with accounting, record-keeping and reporting requirements very similar to any other large business. Besides removing a lot of clerical work, the use of computers for accounting, planning and management provides advantages that are not possible with manual systems. In many cases computer systems are faster, more cost effective and more efficient.

There are a multitude of other tasks which relate specifically to schools: cafeteria accounting, maintenance, scheduling, pupil census, film reservations, honour roll, school bus scheduling, pupil health records, grade reports, posting grades, test results and library holdings. Computers are now being used to automate many of these tasks.

An important application not yet mentioned is that of class scheduling. This involves correlating the requests for courses supplied by students with the courses offered and the facilities available in the school to plan which teachers will teach which courses in which rooms at what times, and which students will be in each class. Reports are needed for students, teachers and the principal for reporting to higher levels in the school district. With the aid of the computer this information may be obtained rapidly and inexpensively without being so prone to "human error".

2.3 Computer Assisted Instruction Systems

In Computer Assisted Instruction (CAI), the goal is to replace the traditional student-teacher dialog with a student-computer interaction. Because machines communicate faster than humans, they should be able to provide more, and better, communication. This objective has been achieved in drill and practice, and in other narrowly defined circumstances -- for example, classes for sales personal, aircraft and weapon-systems training. However, these circumstances represent only a fraction of all educational environments. In the vast majority of cases, CAI has yet to show results comparable to those associated with an experienced human teacher, even when exceptional programming material has been available [SUGARMAN, 1978].

Instructional material prepared for CAI systems may be compared to a series of textbooks that have been broken down into small units of information. Unlike a textbook, however, the material may be presented to students on many levels of difficulty and each student progresses through the material on an individual path charted by his previous performance. The communication between the student at the terminal and the teacher is private. Each student progresses at his own speed and level, competing only with himself -- not with other students. Fast students proceed more quickly because they make fewer mistakes, while slower students take longer because they are informed of their mistakes and are allowed to try again.

CAI systems are currently being used in many educational environments: (1) At the University of Illinois most courses are using CAI instruction in chemistry, physics, foreign languages, medicine and accounting [SUGARMAN, 1978]; (2) At the University of California at Irvine complete courses in physics, mathematics and anthropology are using CAI instruction; (3) At Stanford University four complete mathematic courses and parts of several foreign language courses use CAI; (4) and at the University of British Columbia NATAL-74, a typical CAI language, is currently being used.

Perhaps the most significant advantage of CAI over more traditional instruction is its flexibility. In a typical CAI classroom you might find one student doing drill on Spanish verbs, another student conducting a chemistry experiment, yet another student solving a mathematical problem and other students studying French grammar, English, history and physics. Each student progresses at his own learning rate -- the one which is best suited for him.

Often when a person thinks of drill and practice materials, he visualizes long sequences of addition problems or vocabulary words and associates such exercises with dull repetition, rote memory, low incentive and few side benefits.

This need not be the case. There are several examples of well-designed drill materials in elementary mathematics [DUGDALE, 1975] and one in science [SMITH, 1971]. Each of these examples exhibits more than just rote memory practice. First, motivation is provided by making a competitive game of the practice material. The practice comes as the student generates (by addition, subtraction, multiplication and division) many numbers in determining what he thinks is his best move. The student competes against time, the computer, or another student. In addition to reinforcement by drill, another objective of the material is practice in decision making and strategy development. The student must make decisions as to how to proceed in the game and as he makes these decisions he develops a strategy that helps determine the outcome of the game. This type of drill and practice gives up some of the control needed to measure the type of difficulty a student is having, but it has many additional advantages.

Of all the advantages claimed for CAI, the most important is that it enables each student to learn at his own pace instead of being forced to stay locked in step with classmates whose knowledge and learning rates are different. Individualized instruction has long been a goal in education [WEISGERBER, 1971].

Papert emphasizes the versatility of CAI and the exciting opportunities it offers for teaching concepts and techniques in mathematics, biology, and mechanics even to elementary school students [PAPERT, 1971].

To provide the reader with a feeling for the current state of the art in CAI systems I present a brief survey of five recent research projects: SOPHIE, PLATO, EUCLID, PILOT and NATAL-74.

These five projects were selected because they are representative of the CAI systems currently available and they illustrate the various themes which CAI researchers follow in their systems' design and implementation. In particular, SOPHIE is representative of the artificial intelligence approach towards limited, specific "intelligent" CAI systems. PLATO represents the more or less traditional general-purpose large scale approach. EUCLID represents the traditional special-purpose approach. PILOT is representative of systems which are designed for minicomputers and small-scale systems. Lastly, the course ware for NATAL-74 exemplifes the various author languages available for CAI. This system is a direct descendant of the first widely promoted author language, COURSEWRITER, which was developed by IBM in the early and mid-1960's.

2.3.1 SOPHIE

SOPHIE is an instructional system for teaching expert-grade electronic troubleshooting which has been under development at BBN for several years [BROWN, 1974 and 1976]. SOPHIE was developed to provide symbolic knowledge, problem solving strategies and natural capabilities similar to a human tutor. In particular SOPHIE uses many of the concepts and techniques of Artificial Intelligence. SOPHIE is especially important because it departs from the traditional CAI approach of providing only a question-response environment -- it is able to draw conclusions based on its "intelligence". The primary seat of this intelligence resides in a collection of special purpose inferencing procedures, each of which performs a certain class of inferences extremely efficiently. The most important of these procedures drive special and general purpose simulators in order to determine the result of some particular action. The factual data base resides in a semantic net.

2.3.2 PLATO

Beginning in 1960 as a single terminal, which consisted of a small keyboard and an ordinary television receiver connected to ILLIAC I, PLATO has evolved over the past 19 years to the PLATO IV System. This is probably the largest, most heavily-funded Computer Assisted Instruction experiment in the world [DENENBERG, 1978]. Bitzer and Braunfeld at the University of Illinois initiated the PLATO project. The system was used in its early stages for teaching computer-related topics, as well as mathematics and language drills [KINGERY, 1967].

The PLATO system is based on a Control Data Corporation Cyber 73-2 computer with a high-speed central memory of 65,000 60-bit words and two central processing units each capable of approximately one million instructions per second [BITZER, 1976].

PLATO is constantly being extensively evaluated by the Educational Testing Services at Princeton, New Jersey. The studies are on the use of PLATO in a variety of educational settings.

PLATO has several aids to help students, teachers and evaluators in gathering and processing relevant data. First, the author or teacher can specify what variable or computer-student transactions should be stored by the computer for later processing. Second, the stored information can be sorted, processed and displayed in any manner specified by the teacher. The security of the system makes it possible for a student to see such things as his progress through the lesson material, test scores and relative standing in the course, without being able to see the identical results for other students.

The PLATO computer-based educational system has been specifically designed to provide interactive, self-paced instruction to large numbers of students [BITZER, 1976]. Lesson material is displayed on a screen 22 centimeters square and may consist of text, drawings, graphs, and colour photographs. Students interact with the material through a special keyset that closely resembles a typewriter keyboard, and they receive essentially instantaneous reinforcement of correct work and assistance when they encounter difficulties. Students can work at their convenience in classrooms. The users of PLATO range from grade school students learning reading and mathematics to graduate students in the medical sciences [SMITH, 1976]. The various PLATO installations located in universities, colleges, community colleges, public schools, military training schools and commercial organizations now include several different mainframes and over 950 terminals. The PLATO system at the University of Illinois provides users access to more than 3500 hours of instructional material in more than 100 subject areas [SMITH, 1976].

A complete lesson on PLATO has many of the characteristics of a chapter in a textbook. Like chapters in a book, such lessons need to be assembled in a form that is easy for students to use. All of the lessons associated with a particular course can be made available from an index. Then, when a student signs on to the PLATO system with his name and the name of the course, he may choose topics or lessons to study. He makes his selection from a list of descriptive titles, much as chapters in a book are selected from the table of contents.

There are three main components of the PLATO aspects of the course: instructional lessons, homework and an on-line gradebook. Students are assigned instructional lessons to study; a part of their grade is based on how many of these lessons they complete. Homework is graded by PLATO rather than by the instructors. The student is given printed homework problems that he is encouraged to work at home. When the student is ready, he goes to a terminal to enter his results. Each instructor can look at and change the scores of his own students and can see scores for his section marked on the course distribution graph. An important benefit of this machinery is that an instructor can plan class activities on the basis of up-to-date information on how far students have gotten in their studies.

PLATO has been integrated into instruction in many other ways. For example, in language courses (including French, Spanish, German, Russian, Hebrew, Latin and Esperanto), PLATO is used heavily to drill the student on vocabulary and grammar and to give practice in translating sentences from one language to the other. Another important integration of PLATO into courses is illustrated by the use in chemistry of simulated laboratory experiments as a means of better preparing the student for a real laboratory experiment.

The PLATO computer-based teaching system provides individualized instruction to hundreds of students simultaneously. Lesson material can include natural language dialogues, graphics, numerical and algebraic exercises and colour photographs.

2.3.3 EUCLID

EUCLID is a tool for individual instruction in high school geometry along more or less traditional CAI lines [KELANIC, 1978]. With EUCLID students work at their own rates and along their own routes in constructing proofs to theorems. EUCLID is the fourth version of a theorem-proving computer program written by Kelanic since 1972. The present version was developed at Taylor Allderdice High School, Pittsburgh, Pennsylvania, in 1977.

EUCLID is written in extended BASIC. Various segments contain the general logic of interactive proofs, a Turing controller and a user command interpreter. The Turing controller provides for fully automatic theorem-proving by determining which step to take next. The user command interpreter operates on all input strings having fewer than four characters.

Program segment EUCLID offers the user one of four modes of operation. In all four modes, the user must input the given assumptions and the statement to be proved. The four modes are as follows: (1) Demonstration Mode: The Turing Controller directs EUCLID in an attempt to prove the theorem automatically. The program supplies all statements, reasons, conclusions, and hints to itself. The user sits back and watches the changing file status message. (2) Practice Mode: EUCLID supplies all reasons, all conclusions and permits user access to hints and conclusions. The user must supply all statements and all reasons. (3) Quiz Mode: EUCLID supplies all conclusions, checks user reasons and permits user access to hints and conclusions. The user must supply all statements and all reasons. (4) Test Mode: EUCLID checks the reasons. User access to hints, conclusions and file status is denied. The user must supply all statements, reasons and conclusions.

2.3.4 PILOT

PILOT is a dialog-oriented interactive language for use by teachers and students on small systems [YOB, 1977]. Its simple syntax and free format encourage innovation and use by those frightened by computers or who lack time to learn a more complex language.

PILOT was developed in 1969 at the University of California Medical Center by John Starkweather to meet some of their instructional needs. It was used to train students in pharmacology and later in an elementary school in Marin County. Stanford Reasearch Institute used PILOT in an experimental educational research project (with very good results) and later developed a dialect, called PYLON [YOB, 1977]. In 1971 and 1972 other variants of PYLON were developed by Stanford University, the California State College computer network and the Lawrence Hall of Science at the Berkeley campus of the University of California.

PILOT is a simple language which is entirely word- and dialog-oriented. It is easier to write a simulated dialog in PILOT than in most other languages. PILOT has several features which aid the creation of dialogs. It has minimized the syntax that often confuses the word-oriented human programmar.

2.3.5 NATAL-74

In 1974 the National Research Council and IBM completed the design of a language for CAI called the NATional Author Language, NATAL-74 [WESTROM, 1974].

NATAL was implemented because IBM and NRC thought that CAI would be an important factor in improving education in Canada. The system was first implemented on the NRC PDP-10 computer in 1976 in Ottawa. The NATAL-74 system is now operational in Ottawa and currently is being implemented at UBC in the Department of Education.

NATAL contains three subdivisions -- procedure, unit and a function. The procedure is the routine which controls or manages the lessons. Units are invoked only by being called from procedures. Each unit invocation is a three step process. First, the procedure must determine which unit is to be invoked and assembled and provide all data needed by the unit for its execution in the calling parameter list. Second, the unit is called. Third, the results of the unit are worked on if necessary and stored away for future use in selecting other units.

The unit is the basic routine from which all lessons are constructed in the NATAL-74 language. All interactions with the student must be specified in each unit routine.

Units may call the graphic function for the presentation of special displays, the font block function to change the display character set, the edit function to edit students' responses and the general function to perform response analysis. Units may not invoke other units, nor may units be implicitly linked together, but they may be linked by adjacent calls from a procedure.

There can be only one response statement in a unit, but if more are needed there may be a display statement preceding each response.

2.3.6 A Brief Comparison of the Systems

Each of the five systems presented has notable strengths and weaknesses. These are due in part to their original design, but also may be seen as resulting from the fact that they do not all have the same objective. In particular, the PLATO language and system is the only one with a long history of success in general CAI. NATAL-74 is attacking the same area, general CAI, but it is too early to evaluate its success or failure since experience with it is still quite limited.

SOPHIE is an experimental vehicle for testing some specific artificial intelligence techniques. It has limited applicability, but the artificial intelligence techniques may become very valuable in implementing future CAI systems.

EUCLID is representative of specialized systems. It was built to provide CAI in a single domain and exhibits success in this area. It is distinguished from the other systems (except SOPHIE) because it incorporates a theorem prover within its basic design.

The PILOT system is distinguished from the others in that it implements CAI in a small scale system environment. A variety of constraints are imposed such as limited memory, limited auxiliary storage, a small number of users, and the like, which affect the system design. The most important such restriction is PILOT's reliance on dialogue rather than interaction formats.

2.4 Programming and Problem Solving

Perhaps the most exciting application of computer technology in the curricula of today's secondary schools is in the area of problem solving -- the use of a computer to assist in the study of problems in an existing non-computer curriculum. The computer is most frequently used in mathematics, chemistry, physics, science and business. Problem solving may be thought of as curriculum-oriented rather than computer oriented. The problem solving method is not new; it is as old as man. The use of the computer in the problem solving process is relatively new; it creates excitement, encourages creativity and makes the subject matter more interesting to the student.

The problem solving approach is based on recognition of the fact that teaching is not telling. Problem solving is not merely the communication of knowledge or the learning or memorization of facts -- it is discovering new things, thinking, analyzing and searching. Students are confronted with problems which are to be solved on a computer. They begin to think. They conceive method for solving a problem. They then obtain data and arrange that data in such a way that it is meaningful. Problem solving forces the student to carefully interpret both problem and solution.

3. LOGO in Perspective

3.1 Brief History of LOGO

The development of LOGO began in 1968 at Bolt, Beranek and Newman, Inc., funded by a grant from the National Science Foundation, in collaboration with Seymour Papert, a consultant from the Artificial Intelligence Laboratory at MIT. Since its introduction in Navy classrooms for teaching young children, LOGO has been used in two Massachusetts schools: Anderson Field Elementary School (for a fifth grade class), and Emerson grammar school [FEURZEIG, 1978].

Since then LOGO has continually been improved and it has been used in several research projects dealing with children and their desire to learn.

3.2 Use of LOGO

3.2.1 Bolt, Beranek and Newman, Inc.

BBN has done considerable work in using LOGO to teach mathematics. Much of their work is documented in a multi-volume report [FEURZEIG, 1971].

The BBN effort strongly emphasizes teletype geometry and the use of LOGO to introduce geometric concepts. A variety of experiments have been performed by BBN, primarily in the Boston metropolitan area, to investigate LOGO's utility.

BBN has developed teaching sequences on geometry, number representation, arithmetic algorithms, functions, equations and on the conversion of story problems into formal mathematical terms.

Much of their work aims to provide aspiring mathematics teachers with the skills needed to teach problem solving.

3.2.2 Artificial Intelligence Laboratory at MIT

In the past few years, the MIT LOGO group, directed by Papert, has employed computers to provide a creative learning environment in a public-school setting, has designed new course material that allows graphical interactions in high-school mathematics and physics, and has augmented the sensory environment of handicapped children [SUGARMAN, 1978].

Much of the MIT work has been directed towards children and their experiences using LOGO and turtles.

One research project dealt with the development of computers and computer controlled devices [PAPERT, 1971]. The goal was to achieve a more involved and individual participation of children in their school work. Using LOGO and turtle the children were able to achieve personal goals and improve in their school work. Another research effort dealt with the development of critical thinking; in particular it was concerned with teaching children to analyze how things perform the way they do [PAPERT, 1970]. With the aid of the turtle the children started to understand that pushing a button marked "forward" also made the turtle move forward.

Many other uses of LOGO, including the use of the turtle, music notation, drawing pictures and carrying out tasks, have been documented in MIT reports. Additional topics include children solving mathematical problems, games and LOGO itself. One especially significant project dealt with preschool children learning to communicate with and program the turtle

[PERLMAN, 1974]. This experiment consisted of several LOGO control boxes designed so that only a few new concepts were introduced at a time, and more were added when the child became familiar with what he was doing.

The first box the child saw was called the command box. It had nine buttons. Each button evoked an immediate response from the turtle. Next, when the child mastered the first box a larger command box was presented. This box consisted of the nine buttons from the smaller box plus an additional eleven.

The third box was called a memory box. This box had four buttons controlling the actions: START, STOP, DOIT, FORGETIT and HONK. This box allowed the students to create turtle programs.

One of the main qualities needed to help a child play with a button box is "common sense". That is why the child is shown only a little bit at a time, more buttons being added when the child is sure of the previous set. It is important that the child never feel that he is "too dumb" to learn or that the button box is "too hard" for him. It is important not to propose problems which the child cannot solve. Thus the child is confident of his ability at all times.

3.2.3 University of Edinburgh

The University of Edinburgh has done a considerable amount of work on using LOGO and turtles.

One research report describing an aspect of an experimental project being undertaken at the university is especially noteworthy [duBOULAY, 1977]. The project's attention was directed at the learning and understanding of elementary mathematics of two small groups of student teachers. These student teachers were volunteers from a local college near Edinburgh where they were taking a three-year diploma in primary education. The course work devised for these students was an example of the use of computers in higher education. The main commitment of the project was to explore the effect on each student teacher of her interaction with a special learning environment based on LOGO.

The objectives of the project were to have the student teachers become aware of two main areas that related to teaching mathematics in a primary school: understanding what is being taught and understanding how to teach.

Another research project investigated how the fascination for machines shown by autistic children could be exploited in a LOGO-based learning environment [WEIR, 1976]. The project was about an experience with a seven-year-old autistic child whose active and enjoyable explorations in controlling the LOGO turtle formed the basis for the development of language for communication, both verbal and nonverbal. The researchers discovered two features not shown by the child before. One was the onset of spontaneous language based on descriptions of the turtle's behavior and the second was the active seeking out of social interaction. This discovery, the authors believe, follows from the self-validating effect of understanding and being understood. This in turn follows from the highly structured but creatively open-ended nature of the LOGO environment in which the crucial step of "seeing what is relevant" is made transparently easy.

Earlier work by Omar Moore in the nineteen fifties [MOORE, 1964] showed a similar effect with the "talking typewriter". Computers are not, therefore, necessary for this effect to be demonstrated. However, they do allow for the flexible use of machine slaves. A well designed toy train layout could conceivably work just as well, but less flexibly.

3.3 Competing Languages

The thrust of computers in elementary education seems to be mostly in four areas: first, courses designed to teach about computers, how they function and how to program them; second, using the computer to pass curricula or other student materials to the student; third, actively involving the students in using the computer to solve problems in their course to study; and fourth, allowing students to use the computer for their own expression, self-integration and growth.

Each of these areas have languages associated with them. FORTRAN, Assembly Language, COBOL and BASIC are taught with courses concerned with how to use and understand the computer. PLATO and the IBM's Coursewriter are both CAI languages which help students learn materials for their individual needs. Most problem solving is done in BASIC with small efforts in FORTRAN and LOGO. PILOT and LOGO are often used for self-expression and growth.

BASIC is in essence FORTRAN with many non-essential constructs removed. Its ready availability on small systems (especially time-sharing systems) makes it quite popular for mathematics and science teachers. The student learns BASIC as a tool for solving numerical problems posed in his courses. However, BASIC is very weak with strings, words and data structures; this may account for the fact that BASIC users tend to be in the scientific and technical disciplines.

LOGO is a simplified variant of LISP in a manner analogous to the way that BASIC is a simplification of FORTRAN. Its primary advantage over LISP for CAI purposes lies in its capability for controlling devices such as the turtle, a plotter-robot, and a "music box". LOGO is capable of handling lists and recursive function calls. LOGO is excellent for problems concerning the order and arrangement of things rather than calculations. Unfortunately, at present there are few users of LOGO and they are mainly mathematics and computer-oriented.

4. Summary of LOGO

The University of British Columbia's implementation of LOGO, BCLOGO, is an interpreter written in BCPL -- a machine-independent language designed especially for compiler writing [MANIS, 1973]. It currently operates on the Amdahl 470/V6 under the Michigan Terminal System.

Every language has a set of rules and LOGO is no exception. LOGO's syntax is very simple -- the action to be performed always precedes the listing of the data to be used.

LOGO allows you to define your own procedures. There are built-in simple procedures which print sentences or words and procedures for handling input/output. Procedures may be recursive.

LOGO includes a primitive editing system as well. Along with the Edit command there are Show, Erase, Showline and Save. Each of these commands helps you once you are inside a procedure.

BCLOGO is initialized by reading in a file of commands. These commands cause procedures to be defined, error messages to be stored, initialization procedures to be called and messages to be typed to the user.

Since LOGO is considered by many to be a "baby LISP" it may be useful to discuss the differences between these two programming languages.

LOGO introduces two data types -- WORDs and SENTENCEs. Along with the standard list operations of FIRST (CAR) and

BUTFIRST (CDR), LOGO provides LAST and BUTLAST [GOLDSTEIN, 1975]. All four of these operations work on words as well as sentences. Repeatedly BUTFIRSTing a sentence in LOGO always terminates in the empty list. In LISP, with its more general list structure built from "dotted pairs" and CONSing, this is not always so. In LISP programs are themselves list structures whereas in LOGO they are not. Editing and debugging in LISP consequently becomes awkward due to the difficulty in naming parts of the program. LOGO simplifies program structure by requiring that a program be a series of numbered lines. The locations of bugs and intended edits are then far easier to describe. On the other hand dynamically creating programs in LOGO is very awkward. DO (the interpreter corresponding to LISP's EVAL) is consequently rarely used explicitly by the user.

LOGO's lack of canned loops such as DO and MAPCAR can be criticized as encouraging bad programming practice, such as excessive use of GO. This obscures the logical structure of programs. Also, it may be significantly confusing to the beginner and the source of many bugs. A child might understand quite well a control structure concept like "do this part of the program three times", or "do this part of the program for each element of the list", but may be unable to express that control structure in terms of jumps and conditionals. LOGO programs can't be automatically formatted to reveal their logical structure as can programs written in LISP or block structured languages.

A complete description of the syntax and semantics of LOGO is provided in the LOGO User's Manual [POLLACK, 1978].

5. Teaching Materials Built Around LOGO

I have seen children turn away from school because of the lack of interest or the inability to read and understand what they are reading. For this reason I choose to help children with the aid of the computer. For if a child enjoys his work then he will work harder to achieve more. My interests lie in the elementary levels and for this reason my programs and the manual are aimed at that level. I picked the LOGO language because to me it showed enough flexibility for my usage. LOGO is a easy language to learn and easy to understand.

I started out by writing arithmetic programs: one program for each arithmetic operation -- addition, subtraction, multiplication and so on. Next I wrote some spelling programs. One was called "Old MacDonald". In this program you had to type in the name of an animal and the sound that animal makes. To accomplish this the child had to be able to read and understand what was going on. There were other spelling programs which checked to see if you were spelling the words correctly. There were also story programs which asked questions; the answers were inserted into a story which the child had to read and understand. These are amusing programs, but also educational, for they asked the child to use the skills he already has and improve on them. Moreover, the programs would be available to the children to inspect as non-trivial LOGO programs. They would be encouraged to change the programs to produce different behavior. Here are some examples of the programs I wrote. The underlined words are what the program types out and the words not underlined are what the user typed in.

```
= MATHS
TYPE IN TWO NUMBERS
5
6
WHAT IS 5 + 6 ?
10
WELL LET US SEE IF YOU ARE CORRECT
5 + 6 =
11
LET US TRY AGAIN
GIVE ME THE SAME NUMBERS YOU GAVE ME BEFORE
* * * * * * * *
5
6
WHAT IS 5 + 6 ?
11
WELL LET US SEE IF YOU ARE CORRECT
5 + 6 =
11
GREAT -- YOUR ANSWER WAS CORRECT
* * * * * * * *
TYPE IN TWO NUMBERS
0
0
* * * * * * * *
* * * * * * * *
THIS IS ALL THERE IS FOR ADDITION -- GOODBYE


= FARM
OLD MACDONALD HAD A FARM EEI EEI O
AND ON HIS FARM HE HAD A
PIG
EEI EEI O WITH A
OINK
OINK HERE, AND AN OINK OINK THERE HERE AN OINK
THERE AN OINK EVERYWHERE AN OINK OINK
OLD MACDONALD HAD A FARM EEI EEI O
AND ON HIS FARM HE HAD A
..   ..   ..
```

For further information see the the LOGO User's Manual [POLLACK, 1978].

6. Experience in Computer Science 200

At the University of British Columbia in the Spring of 1978 I was fortunate to be given the opportunity to have my Manual used by eleven full-time students in Computer Science 200, Computers and Man, instructed by Dr. Abbe Mowshowitz. The course is designed for non-Computer Science Majors. The eleven students were all from the Faculty of Science ranging from Mathematics to Biochemistry and from first year to fourth year standing.

I was allowed three hours with the class. One hour was spent on an a introduction to basic mechanics: signing on to the IBM 3270 terminals and the Vucoms, running LOGO and answering questions about the terminals, the language and other things. The second hour was spent talking about LOGO -- explaining the uses of the glossary, giving examples using LOGO and discussing the two programming assignments which I presented. The third hour was given three weeks later during which the students asked me questions about LOGO and the manual. During this hour I also handed out a questionnaire which covered the Manual and LOGO.

6.1 The Two Assignments

The two assignments which were given the class were:

(1) Write a procedure which adds up each number you type in and at the end of the procedure the sum of the numbers is printed out.

(2) Write a procedure which reads in a string and reverses that string. Write another procedure which inputs two strings and sees if they are both the same. Combine both procedures to see if a string is a palindrome (that is, it reads the same from front to back and back to front).

6.2 Two-Part Questionnaire

The following questionnaire was used to help evaluate the utility of LOGO and my manual. It was presented to the CPSC 200 students at the end of the term.

6.2.1 Questions on LOGO

The following are questions on LOGO.

1) Is LOGO a good language to learn if you
   do not know any other programming language?

   Very Good Good OK Bad Very Bad

2) If your program has an error in it, does LOGO
   explain the error well enough for
   you to correct it?

   Very Adequate Adequate OK Inadequate Very Inadequate

3) Were there any specific things about LOGO
   you did not like?
   Why?

4) How can the language LOGO be improved?

5) Would you prefer to learn and use LOGO or
   some other language? Specify the language:
   Why?

6) Would you have liked to spend more
   time in this course programming in LOGO?
   How much time?
   Why?

7) If you had the time available, would you like
   to continue working with LOGO and
   the computer?
   Why?

6.2.2 Questions on the Manual

The following are questions on the manual.

1) Does the manual provide enough information to make you a reasonable programmar in LOGO?

   Very Adequate Adequate OK Inadequate Very Inadequate

2) Were you bothered by the fact that the manual was directed at the secondary level?

   Very Bothered Bothered OK Not Bothered

3) Are the number of examples in the manual adequate?

   Very Adequate Adequate OK Inadequate Very Inadequate

4) (a) The best written section of the manual was?
   Why?

   (b) The worst written section of the manual was?
   Why?

5) More additional work is needed in the following areas of the manual.

6) What should be changed or added in the manual to make it easier for you to learn LOGO?

6.3 Questionnaire Results

The results were quite good. I was fortunate to have had students who were willing to spend the time to discuss in detail the good and bad parts of both the manual and LOGO.

Following are the responses of the students to each of the questions found in chapter 6.

Questions and responses on LOGO.

(1) Is LOGO a good language?

Very Good	Good	OK	Bad	Very Bad
3	3	4	1	

(2) Does LOGO explain the errors well?

Very Adequate	Adequate	OK	Inadequate
1	2	4	4

(3) Where there any specific things about LOGO you did not like? Why?

Too expensive
cannot rename a procedure

(4) How can the language LOGO be improved?

Sum and difference be substituted by + -
allow for subroutines
more built in functions
error messages need improvement

(5) Would you prefer to learn LOGO or
some other language?

- 5 for LOGO -- easy to use, to learn, OK to start with but need a more sophisticated language to do more complicated problems
- 3 for FORTRAN -- used for scientific work, but LOGO nice for fun, LOGO easy to learn for a first language
- 3 with no comments

(6) Spend more time with LOGO?

- 6 stated Yes -- more problems, 3 or 4 more weeks, a interesting language, more lecture time
- 5 stated No -- more practical applications would be appropriate, cannot afford to spend the time, not enough time to spend on programs

(7) If you had the time would you like to
continue with LOGO and the computer?

- 8 stated Yes -- fun, useful to learn another language, more difficult programs, experiment with the language
- 3 stated No -- other interests, need a language to solve mathematical problems

<u>Questions and responses on the Manual</u>.

(1) Does the Manual provide enough information?

Adequate	OK	Inadequate	Very Inadequate
1	5	3	2

(2) Were you bothered by the fact that the
Manual was directed at the secondary level?

Very Bothered	Bothered	OK	Not Bothered
		6	5

(3) Are there enough examples in the Manual?

Adequate	OK	Inadequate	Very Inadequate
1	1	5	4

(4) (a) The best written section?

Sign On and Off, editing, how to use and write programs, everything, explaining the interactive nature of the language

(b) The worst written section?

Glossary, :a: as a mailbox is unclear, test and editing sections, how to get out of Edit, recursion examples not good, building programs from commands, whole Manual lacking in sufficient detail to be very useful

(5) Additional work is need?

Glossary, functions, test and editing, more examples through the Manual, show how the commands are to be used

(6) What should be added or deleted?

More complicated examples in mathematical notation, better use of procedures, more examples using Equalp, more examples of programming

I used the results of the questionnaire to help prepare the current version of the manual. I rewrote the glossary and inserted examples for each. I also went through the manual and put in examples where they were needed.

7. Critique of LOGO

Like most systems, LOGO has several notable weaknesses as well as its advertised strengths.

LOGO's weaknesses are that it is limited in its ability to perform numeric computations, and it provides only very primitive control structures and data structures.

The comments of the CPSC 200 students were quite useful in correcting the User's Manual and in helping me evaluate the language. With the variety of backgrounds of the students taking the course it was difficult to form an overall impression of their responses. Some of the students took the course because they had a slight interest in computer science; others had FORTRAN and/or BACIC experience and found LOGO to be redundant and limited; still others were very interested in exploring LOGO's possibilities.

Various suggestions for improvement of the BCLOGO implementation include: enhance the efficiency to cut its cost; provide for renaming of procedures; allow subroutines; provide for the use of infix notation in addition to the current prefix form; improve error diagnostics and implement more graceful error recovery; and add a floating point data type.

LOGO's strengths are: 1) it is an extremely simple language with relatively uncomplicated syntax; 2) it is easy to teach and easy to learn; 3) it is designed for novices; 4) it is interactive; 5) it was designed with the idea that using a computer should be "fun" -- not "work".

Comparison of BASIC and LOGO:

BASIC:

a) The Goal of BASIC: The idea behind BASIC was to make it easy for people and computers to interact. BASIC was developed in the early sixties to provide facilities for people to obtain solutions to mathematical and business problems with the aid of the computer.

b) The Audience: The people who use BASIC are primarily from the business world, i.e., industry, banking, manufacturing and related fields. Additionally a large number of school administrators and students are now using the language. Many scientists in areas other than computer science use BASIC for simple numerical computation.

c) Syntax: BASIC can be used to express problems and their solutions in a form that is readily understandable by humans -- it looks reasonably similar to high school algebra. Some versions include extensions for performing string processing.

d) Implementations: There are many different versions of BASIC being used. One popular version is BASIC-PLUS. BASIC-PLUS allows both recursive functions and recursive subroutines. This version of BASIC uses the IF ... THEN ... ELSE structure, string functions and dynamic string arrays.

e) Summary: BASIC is not a powerful language. It is weak in control structures and weaker in data structures. It is a language meant for interactive computing by novices, primarily in the numeric domain.

LOGO:

a) The Goal of LOGO: The idea behind LOGO was to create a programming language to do practical things with the aid of computers and to describe algorithms. LOGO was designed to work with symbols rather than with numeric values. This feature distinguishes LOGO from BASIC and FORTRAN.

b) The Audience: The people who use LOGO are children with special learning problems, "normal" children and novice programmars.

c) Syntax: LOGO has very few syntax rules. The user's primary goal is to learn what the various features of LOGO do, then how to write them down. LOGO is a interpreted language. This means that LOGO operates in a direct execution mode [MANIS, 1973]. As the user types in a line of code which is not part of a procedure definition, LOGO obeys that command immediately; but if a procedure is being defined, LOGO will store the code away for later use.

d) Implementation: There are many versions of LOGO: Edinburgh, BBN, MIT and UBC all have different LOGO implementations, but they are similar in their semantics. UBC does not have a turtle or a graphic environment working, but the other three do have at least a working turtle.

Many of the CPSC 200 students complained that BCLOGO was costly to use, especially in comparison with other languages such as FORTRAN. There are two reasons for this: 1) LOGO is implemented as an interpreter and is therefore inherently more expensive to run than compiled languages such as FORTRAN; 2) the BCLOGO implementation in BCPL is inefficient. There is no reason inherent in the LOGO language that should make it any more or less costly to use than any other interpretive language (such as BASIC or LISP).

e) Summary: LOGO is good at processing information but it is not particularly suited to "number crunching". But given a program written in BASIC by an elementary or secondary school student to perform some calculation, LOGO could do the job as well as BASIC so long as the numbers are small. However, UBC LOGO has no floating point data type, so the problem would have to be solvable using integers.

The Semantics of BASIC and LOGO:

Both languages have the roughly same format. Examples of some of the most similar statements are:

```
      LOGO                           BASIC

PRINT "this is a test"    PRINT "this is a test"
PRINT sum 8 22            PRINT 8 + 2
MAKE "A" yes              LET A = yes
GO 100                    GOTO or GO TO 100
END                       END
BYE                       BYE
MAKE "X" request          INPUT X
TEST is A yes             IF A = yes THEN 100
IFTRUE GO 100
IFFALSE "wrong"           PRINT "wrong"
```

8. Conclusion

LOGO is an ideal language for the elementary and secondary levels of education, but it is difficult to see how the language could be used in other environments. The primary use of computers is in the business world. Perhaps LOGO should be modified to reflect the needs of this user community. If LOGO were to be used outside of the classroom it would have to be made more powerful in the area of numeric computation. Certainly if the BCLOGO implementation were to be used in production environments the suggested improvements in the previous chapter would have to be made.

LOGO's useful lifespan may already be over -- it has been used in research environments for some time and it may be that it is time to design a successor language, incorporating the knowledge acquired in the various LOGO experiences and experiments.

The future for LOGO is dim unless elementary and secondary school educators are made aware of its potential. Of course, generating such awareness depends on inducing schools to use LOGO. For example, if LOGO were successfully used in one elementary school in the Vancouver school district, others might be encouraged to introduce the language, and perhaps LOGO might find its way into secondary school curricula as well.

Both students and teachers would benefit from the opportunity to select the most appropriate language for their problem-solving tasks. The potential of LOGO as a vehicle for teaching problem solving skills leads one to conclude that a LOGO-like language ought to be provided as an alternative to BASIC, FORTRAN and Assembly Language.

9. References

Bitzer, D. The Wide World of Computer-Based Education, Advances in Computers. Vol. 15 (1976), pages 239-283.

Brown, J., Burton, R., and Bell. SOPHIE -- A SOPHisticated Instructional Environment for teaching Electronic Troubleshooting. Bolt, Beranek and Newman, Inc. Artificial Intelligence, Cambridge, Mass., BBN Report No. 2790 (1974).

Brown, J., Rubinstein, R., and Burton R., Reactive electronics instruction. Bolt, Beranek and Newman, Inc. Cambridge, Mass., BBN Report No. 3314 (1976).

Denenberg, Stewart A., A person evaluation of the PLATO system. ACM Sigcue Bulletin, Vol. 12, No. 2 (1978), pages 3-10.

Dodds, W., Personal communication. Vancouver School Board, Vancouver, B.C. (1978).

du Boulay, B., Learning teaching mathematics. Department of Artificial Intelligence, University of Edinburgh, D.A.I. Edinburgh, Scotland. Research Report No. 18 (1977).

Dugdale, S., and Kibbey, D., The fractions curriculum -- PLATO elementary school mathematics project. Computer-based Education Research Laboratory, University of Illinois, Urbana (1975).

Feurzeig, W., Personal communication, Bolt, Beranek and Newman, Inc., Cambridge, Mass. (1978).

Feurzeig, W., Lukas, G., et al, Conceptual framework for teaching mathematics. Bolt, Beranek and Newman, Inc., Cambridge, Mass., Vol. 1-4, BBN Report No. 2165 (1971).

Goldstein, I., Lieberman, H., et al, LOGO - an implementation of LOGO in LISP. MIT Artificial Intelligence Laboratory, Cambridge, Mass., LOGO Memo 11 (1975).

Kelanic, Thoman J., Theorem-proving with EUCLID. Creative Computing, Vol. 4, No. 4 (July-August, 1978), pages 60-63.

Kingery, R.A., Berg, R.D. and Schillinger, E. H., A computer in the classroom. In Men and Ideas in Engineering: Twelve Histories from Illinois. University of Illinois Press., Urbana, Illinois (1967), pages 147-164,

Manis, Vincent S., A Machine Independent Implementation of LOGO. University of British Columbia, Department of Computer Science, M.Sc. Thesis (1973).

Moore, Omar K., Autotelic responsive environments and exceptional children, in: Jerome Hellmuth, ed. The Special Child in Century 21. Special Child Publication, Seattle, Washington (1964), pages 87-138.

Papert, S., Teaching children thinking. MIT Artificial Intelligence Laboratory, Cambridge, Mass., AI Memo No. 247 (1970).

Papert, S., A computer laboratory for elementary schools. MIT Artificial Intelligence Laboratory, Cambridge, Mass., AI Memo No. 246 (1971).

Perlman, R., TORTIS -- Toddler's Own Recursive Turtle Interpreter System. MIT Artificial Intelligence Laboratory, Cambridge, Mass., AI Memo No. 311 (1974).

Pollack, Kimberly A., LOGO User's Manual. University of British Columbia, Department of Computer Science (1978).

Smith, S. G., Computer-aided teaching of organic synthesis. J. Chemical Education 48 (1971), pages 727-729.

Smith, Stanley G., and Sherwood, Bruce A., Education Educational uses of the PLATO computer system. Science, Vol. 192 (April, 1976), pages 344-352.

Sugarman, R., A second chance for computer-aided instruction. IEEE Spectrum (August, 1978), pages 29-37.

Sugarman, R., "What's new, teacher?" Ask the computer. IEEE Spectrum (September, 1978), pages 44-49.

Weir, S., and Emanuel, R., Using LOGO to catalysis communication in an autistic child. Department of Artificial Intelligence, University of Edinburgh, D.A.I., Edinburgh, Scotland, Reasearch Report No. 15 (1976).

Weisgerber, R. A., Perspectives in Individualized Learning. Peacock, Itasca, Illinois (1971).

Westrom, Marv, NATional Author Language (NATAL-74) Author Guide. National Research Council of Canada, Ottawa, Ontario (1974).

Yob, G., PILOT. Creative Computing, Vol 3, No. 3 (May-June, 1977), pages 57-63.

APPENDIX: LOGO USER'S MANUAL

Dear Sir,

In these fast-changing times with the lowering of moral standards and the increase in new programming languages, I often find troubled young men asking me the question, "Father, how can I be sure that LOGO is the one true programming language?"

For such confused souls and examination of their childhood LOGO catechism often proves of comfort.

Remember how it begins ....

1. What is LOGO? LOGO is a programming language.
2. Who made LOGO? Papert and BBN made LOGO.
3. Why did they make LOGO? They made LOGO to help children and social science students know it, love it and through it learn to juggle and ride a unicycle.
4. Is there only one LOGO? No, there are many LOGO's but they are all reflections of the one true LOGO.
5. Is LOGO just a programming language? No, LOGO is more than a programming language, it is also a theory of learning and the path to deep psychological insight.
6. Will LOGO help me? Yes, LOGO will help you by making your problem solving process explicit and transparent unto you.
7. What are the four last things? Procedures, debugging, problem decomposition and recursion.
8. Will LOGO stop people kicking sand in my face? No, but it should keep you off the beach.

Yours sincerely,
Fr. Aloysius Hacker,

Cognitive Divinity Program
Institute of Applied Epistemology [1]

LOGO USER'S MANUAL

1. Introduction

Welcome to the Wonderful World of LOGO. LOGO is a programming language which is designed to be especially easy to learn. It is designed to teach the elements of programming and problem-solving.

As you read the pages that follow you will be learning a computer language called LOGO. In order to use a computer language you do not need to know how the computer works. You just have to be able to put the information to work for you. LOGO is an advanced computer language designed for children, or anyone who is willing to learn and enjoy the use of a computer. This manual was written for the secondary level of education and hopefully it will be used in helping children with difficulties in mathematics and spelling. No previous experience is needed to use this language or the computer.

The objective of this manual is to bring to the secondary school a different method of using the computer. Currently, the BASIC language is used in many secondary schools in the Vancouver area, but it is my belief that students might benefit more by learning the elements of programming in LOGO, and then moving on to more advanced languages. Certainly students should be made aware of the other programming languages that are around.

2. How to Sign on to a Terminal

At the University of British Columbia you will find the IBM 3270 display station which consists of a television-like screen and a typewriter keyboard. The computer can serve a large number of users concurrently, offering each user a wide variety of services. The job of keeping track of all the programs in the machine and of devoting some attention to each of them every second or two is handled by the MTS operating system. MTS stands for Michigan Terminal System.

When you approach a terminal the screen will read:

```
MTS
```

in large block letters. This simply means that this terminal is ready for your use. The cursor '_' is positioned at the begining of the input area. In the examples everything which is underlined is what the program types out on the screen and what is not underlined is what you type. In order to request service from the computer, you must first identify yourself to MTS. This is done by typing in the word 'Signon' and 4 letters 'XXXX' which are given to you by your teacher. The screen is cleared of the block letters 'MTS' and in front of you the screen should look like this:

```
# SIGNON XXXX
# ENTER USER PASSWORD
?
```

The program is now ready for you to type in your password which will be hidden. Once this is done the '#' mark will be displayed on the screen confirming the completion of the Signon process:

```
                #
```

Now MTS is listening to you and is waiting for your request. This communication is done through the MTS Command Language.

3. How to Run LOGO

LOGO is an interpreted language. This means that LOGO operates in a direct execution mode [MANIS, 1973]. As the user types in a line of code, if he or she is not defining a procedure, LOGO obeys that command immediately; but if defining a procedure, LOGO will store the information away for later use. You should think of LOGO as a friend who is not a very fast learner, but is willing to carry out your orders. LOGO does have one problem it will do what you say, not what you mean. To use this language you simply type in:

```
# $RUN LOGO:LOGO
```

The screen will then come back to you with a '=' sign. This '=' sign means that LOGO is now listening to you. Now you are ready to begin to give LOGO commands to do things for you.

When the time comes that you are ready to leave LOGO you type in:

```
= BYE
```

This will terminate LOGO and bring you back to MTS. To make a copy of your work before you signoff and you are on the VUCOM you press down on the SHIFT key and at the same time hold down the C key and this will print the buffer. If you are on the 3270 terminal you hit the PF3 key and the same thing will occur. If you are still in LOGO and wish to terminate all together without going back in to MTS You type in LOGOUT. This command will ask you if you wish a copy of your work before you are signed off with the charges. If you are talking to MTS and you are through for the day you type in the following command:

```
# SIGNOFF
```

This command will display on the screen the charges for the session and then redisplay the 'MTS' block letters.

4. Running LOGO

LOGO is a very simple language in which to program. You tell LOGO to do something and it will do it. For example, if I wanted LOGO to print out 'HI' I would type in:

```
= PRINT "HI"
```

LOGO would then type out on the next line:

```
HI
```

PRINT is a built-in word. It is an action while 'HI' is the data. Note that the action that is to be performed always must precede the data.

LOGO contains two kinds of objects WORDS and SENTENCES. A word is any string of letters or digits surrounded by quote marks [MANIS, 1973]. A word can have as few as one or as many as 255 characters in it. The following is an example of a word:

```
= PRINT "2345%$"
2345%$
```

A sentence in LOGO is simple a sequence of words separated by blanks. The following is an example of a sentence:

```
= PRINT "HI THERE, LOGO"
HI THERE, LOGO
= PRINT SENTENCE "HI" SENTENCE "THERE," "LOGO"
HI THERE, LOGO
```

By joining sentences of words together you get a LIST. The following are some examples of a list:

```
= PRINT LIST "HI" "THERE"
HI THERE
= MAKE "A" "LOGO"
= PRINT LIST "HI," :A:
HI, LOGO
```

For putting words and lists together, there is the command WORD. WORD takes two inputs, both of which must be words, and puts them together to make a longer word. For example:

```
= PRINT WORD "HI" "THERE"
HITHERE
```

LOGO has four operations and when used with the command word or sentence results in a very interesting output. For example, FIRST outputs the first character of its input if the input is a word; and if the input is a sentence, FIRST outputs the first word. BUTFIRST outputs everything *but* the *first* element of its input. LAST and BUTLAST are similarly defined. The following are some examples of the usage of these operations:

```
= PRINT FIRST "CAT"
C
= PRINT BUTFIRST "CAT"
AT
```

```
= PRINT LAST "CAT"
T
= PRINT BUTLAST "CAT"
CA
= PRINT FIRST "LOGO IS FUN"
LOGO
= PRINT BUTFIRST "LOGO IS FUN"
IS FUN
= PRINT FIRST FIRST "LOGO IS FUN"
L
```

The output of FIRST or LAST is always a LOGO word. The output of BUTFIRST or BUTLAST are always the same as their input - either a word or a sentence. This means that a single word can sometimes be a LOGO sentence. The following are examples of using more than one operation in a command:

```
= PRINT SENTENCE "GO" WORD "MAN" "GO"
GO MANGO
= PRINT LAST (FIRST "THE CAT")
E
```

LOGO has many kinds of instructions which do things for you. One kind of instruction is the COMMAND which is an order to do something like "Pick up your book". An OPERATION, on the other hand, answers a question like "Yes, you can pick up my book."

For both operations and commands, the instruction is written with the name of the desired action, followed by the data that the action needs. The input to an operation or command may itself be the result of another operation, but cannot be a result from a command since commands do not output anything. Here are some examples which you might try:

```
= PRINT "X"
```

means that 'X' will be typed out on the next line.

```
= SUM "5" "4"
```

means that the number '5' and the number '4' will be added together. But if you just type in the line without a command LOGO will return a error message:

```
ERROR MESSAGE
"YOU DID NOT TELL ME WHAT TO DO WITH 5 + 4."
```

The command PRINT needs only one argument. If you use a word with this procedure you need to surround the word in quotes, as shown above. But if the argument is a number then there need not be any quotes. Without the quotes around the letter X, Print would try to execute the procedure X and a error message would be printed. Print also will accept a sentence. All you need to remember about sentences, is that they are composed of a double quote followed by a sequence of words, ending with a double quote. Here are some examples:

```
= SUM "1" "2"
ERROR MESSAGE
"YOU DID NOT TELL ME WHAT TO DO WITH 3."

= PRINT SUM "1" "2"
3

= PRINT HELLO
ERROR MESSAGE
"LOGO DOES NOT KNOW HOW TO HELLO."
```

```
= PRINT "HOW ARE YOU TODAY."
HOW ARE YOU TODAY
```

Now let us use these two words PRINT and SUM and try some arithmetic problems. If you take:

```
(7 + (5 - 2))
```

You know that in arithmetic the result would be 10. You would first take the innermost parentheses and get the result; then with that result you would go to the outer parentheses and solve the problem. In LOGO you would type in the same problem as follows:

```
(SUM 7 (DIFFERENCE 5 2))
```

To have the solution print out on the next line you must use the command Print. So the complete sentence in LOGO would read as follows:

```
= PRINT (SUM 7 (DIFFERENCE 5 2))
10
```

The parentheses are there to help you when you look at the problem, but are not necessary to solve the problem.

In this example the procedure DIFFERENCE is given the numbers 5 and 2. This procedure returns a result which happens to be the number 3 this time around. Then, the procedure SUM is run with the numbers 7 and 3, where 3 was the result from the procedure DIFFERENCE. The procedure SUM also returns a result, which happens to be the number 10. Next the command PRINT takes over. Its effect is to print the number 10 on the screen. Now, let us try another sentence using the same procedures, but in a different order. First let us see what it would look like in arithmetic form:

$$(7 - (5 + 2)) = 0$$

Now let us write it in LOGO form:

```
(DIFFERENCE 7 (SUM 5 2))
```

And with the command PRINT you get:

```
= PRINT (DIFFERENCE 7 (SUM 5 2))
0
```

In this example the procedure SUM is given the numbers 5 and 2. This procedure returns a result which is the number 7. Then, the procedure DIFFERENCE is run with the numbers 7 and 7, where the last 7 was the result from the procedure SUM. The procedure DIFFERENCE also returns a result, which happens to be the number 0. And at last, the command PRINT takes over and it prints the result of these procedures, on the screen.

All the arithmetic procedures need two inputs before they can calculate their results. If you forget to use the command PRINT, LOGO will return a message to you saying:

```
            YOU DID NOT TELL ME WHAT TO DO WITH RESULT
```

where the result is the answer or value your sentence calculated.

LOGO reads your commands from left to right, looking for the correct number of inputs for each procedure. Also note that in LOGO operations always come before the numbers operated on whereas in algebraic notation the operations come between the numbers.

The LOGO command MAKE is used for naming. MAKE takes two inputs; the first is the name and the second is the thing being named.

```
= MAKE "X" 27
```

This statement will assign the name 'X' to 27. A name must be a LOGO word. 27 is the value or thing, and may be a number or a word or a list or anything else you wish it to be.

```
= PRINT :X:
27
= MAKE "MILK" "LIQUID"
   (MILK IS THE NAME AND LIQUID IS THE THING)
= PRINT :MILK:
LIQUID
```

5. Defining Procedures

LOGO allows you to define as many separate procedures as you like, and it stores them all in your section of working space. Because you can have more than one procedure in the working space it is necessary to give each procedure its own name so that you can run the one you want.

Defining a procedure is like telling someone what to do. Your instructions should be clear and easy to follow. For example:

```
HOW TO DRINK A GLASS OF MILK
   1 GET A GLASS,
   2 GET THE MILK,
   3 POUR THE MILK INTO THE GLASS,
   4 PICK UP THE GLASS AND DRINK THE MILK.
```

In order to define a procedure you must [MANIS, 1973]:

a. Think up a name or title for your procedure that will help you in remembering what it does.

b. Decide on what inputs and output you want your procedure to have.

5.1 Examples of Procedures

In the example that follows you are shown how to define a new procedure which will be named GREET. This procedure's job is to output two sentences on the display screen.

```
= TO GREET
@ 10 PRINT "HELLO"
@ 20 PRINT "WHAT IS YOUR NAME?"
@ END
```

We call 'TO GREET' the 'title line', and GREET is the name of the procedure. The command END takes you out of editing mode and it also causes LOGO to note that you have finished with the definition of the procedure, and it will respond with the statement:

```
GREET DEFINED.
```

The body of a procedure is stored in the order of its line numbers. Therefore you should start off at the number 10 and increase by 10. Then, if there are any insertions, you have room for them. Once the procedure is defined it is stored in LOGO's memory until you say 'BYE'. If you wish to be able to run one of your procedures at another session and do not want to have to define it again you can command LOGO to 'SAVE' the procedure. The command SAVE needs only one input. This input should be the quoted name of a procedure which is to be remembered by LOGO. For example:

```
= SAVE "GREET"
```

When you start up LOGO the next day or the next session and you have SAVED the procedure GREET from the last session all you must type in is the following line:

```
= GET "GREET"
```

This line will recall the procedure GREET from LOGO's memory. You must type in this line, with the correct procedure 'NAME', for every procedure you saved and wish to work with. When you do not wish to have the procedure GREET around any more all you do is type in the following command and the procedure GREET will no longer exist:

```
= ERASE "GREET"
```

5.2 Running Your Own Procedures

To run any procedure you must type in the name of the procedure without the 'TO'. LOGO assumes that names without TO are procedures to be executed. Running a procedure is like telling someone to carry out the instructions that you have given him earlier.

To run the procedure GREET you just type in the name without the TO, and the procedure will follow the instructions that you told it to do. For example:

```
= GREET
  HELLO
  WHAT IS YOUR NAME?
```

The procedure GREET's job is to execute the two lines in the right order. Each one of the lines contain the name of one of LOGO's commands. In the case it is the command PRINT, and as you can see two sentences are printed out on the screen.

5.3 Procedures with Inputs

After you have tried running the examples shown above a few times you are probably ready for a new procedure.

A procedure with an input is one which expects to receive information from you, the user. Remember the procedure GREET? Well, let us create a new procedure with its aid.

```
= TO GREET
@ 10 PRINT "HELLO" (PRINTS OUT THE WORD 'HELLO')
@ 20 PRINT "WHAT IS YOUR NAME?"  (PRINTS SENTENCE)
@ END (END OF THE PROCEDURE)
GREAT DEFINED. (THIS IS PRINTED BY LOGO)

= TO RESPONSE :PERSON:
@ 10 PRINT "HELLO" (PRINTS OUT THE WORD 'HELLO')
@ 20 PRINT "MY NAME IS" (PRINTS SENTENCE)
@ 25 PRINT :PERSON: (PRINTS THE NAME GIVEN)
@ END (END OF THE PROCEDURE)
RESPONSE DEFINED. (THIS IS PRINTED BY LOGO)
```

Do you see the difference between these two procedures? There titles are different. In the procedure RESPONSE the word person surrounded by colons as been added to the 'title line'. Note that the word PERSON comes after the title of the procedure. By enclosing PERSON in colons you are telling LOGO that PERSON is a variable. A variable is simply a holding place for a word or number you wish to use at a later time. In other words a variable is simply the name of the mailbox in which values can be stored. For example:

```
= RESPONSE "KIMBERLY"
  HELLO
  MY NAME IS
  KIMBERLY
```

The word 'KIMBERLY' is put in the mailbox named PERSON. So until you change PERSON, 'KIMBERLY' will remain in the mailbox.

You may define a procedure with as many inputs as you want by adding after PERSON another colon with another variable. For example:

```
= TO EXAMPLE :PERSON: :AGE: :SEX: :WEIGHT:
```

The LOGO operation REQUEST waits for the user to type in a list and then outputs that list.

```
= TO AGREE
@ 5 PRINT "TYPE SOMETHING YOU LIKE"
@ 10 PRINT SENTENCE "I" S "LIKE" S REQUEST "TOO"
@ END
AGREE DEFINED.

= AGREE
TYPE SOMETHING YOU LIKE
CAKE AND ICE CREAM
I LIKE CAKE AND ICE CREAM TOO
```

REQUEST always outputs a list, even if it is a list containing one (or no) words.

6. Recursion

A procedure can call itself. This is called a recursion. It can be a lot of fun to use, but it also can cause LOGO to run wild which may be quite expensive. For example:

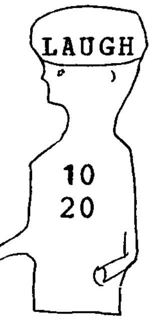
```
= TO LAUGH (to is a command and
            laugh is the procedure name)
@ 10 PRINT "HA HA" (prints ha ha)
@ 20 LAUGH (calls itself)
@ END (procedure stops)
LAUGH DEFINED. (this is printed by LOGO)
```

What you will see on paper is the following:

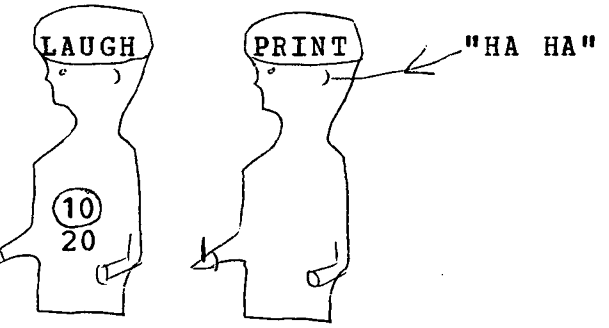
```
= LAUGH ( name of the procedure)
   HA HA
   HA HA
   HA HA
   ETC.
```

There is no way to stop this procedure except to hit the INTERRUPT button.

When the procedure LAUGH reaches line 20 it calls for the execution of the sub-procedure LAUGH, and it will then start a new procedure called LAUGH which will also print HA HA's. The result is a big loop, and page after page of HA HA's. The example that follows shows a step by step trace through the procedure LAUGH.



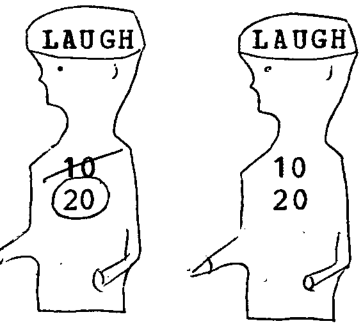
Snapshot 1



LAUGH calls PRINT

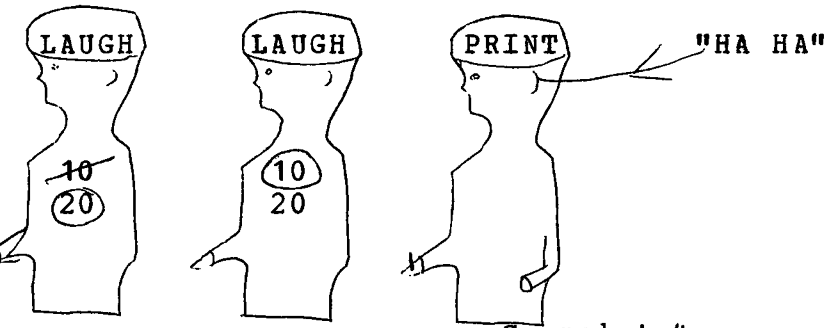
HA HA

Snapshot 2



LAUGH calls another LAUGH

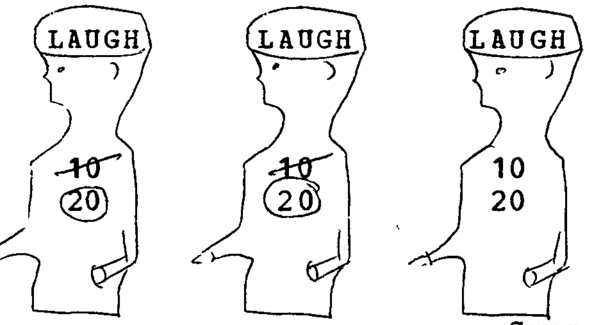
Snapshot 3



The new LAUGH calls
PRINT

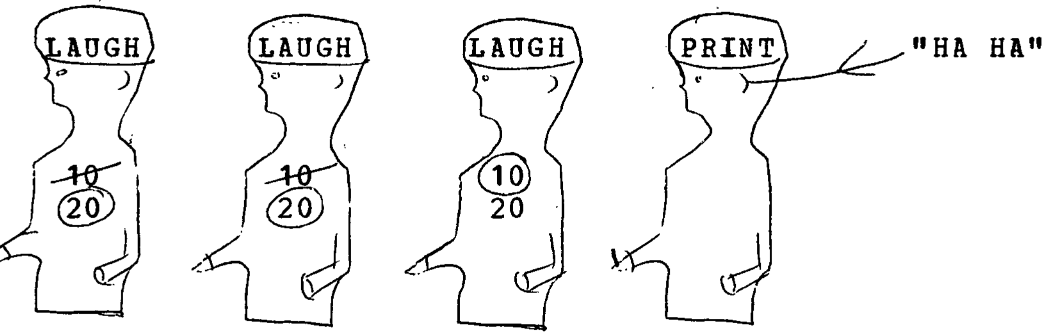
HA HA
HA HA

Snapshot 4



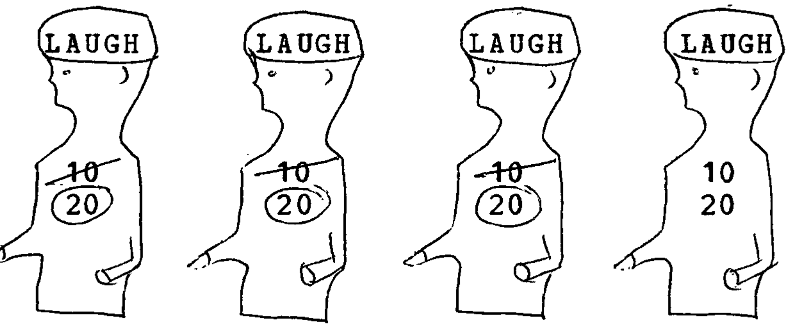
THE new LAUGH calls
yet another LAUGH

Snapshot 5



```
HA HA
HA HA
HA HA
```

Snapshot 6



Snapshot 7

So please remember, although a recursive procedure may be fun to try, it may also be very expensive. Make sure you are able to stop the procedure once it has started to run.

Now, to try a recursive procedure that will stop. To do this you must put a limit on how many HA HA's will come out. Therefore you must have a procedure with an input. For example:

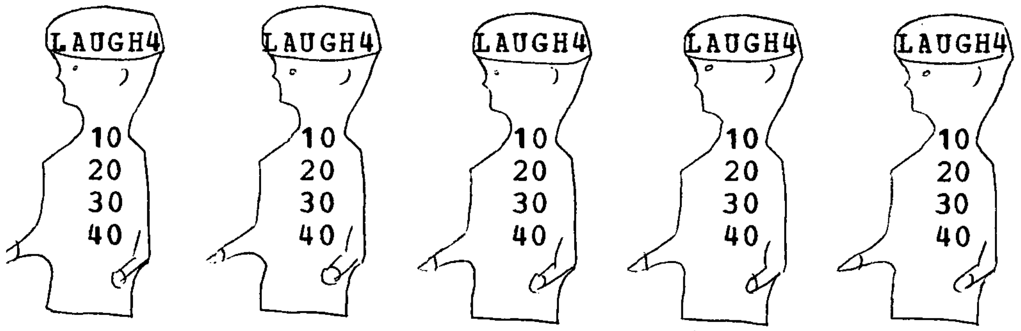
```
= TO LAUGH :N:
@ 10 TEST IS :N: 0
@ 20 IFTRUE STOP
@ 30 IFFALSE PRINT "HA HA"
@ 40 IFFALSE LAUGH DIFFERENCE :N: 1
@ END
LAUGH DEFINED.
```

Now, let us run this new procedure 'LAUGH'.

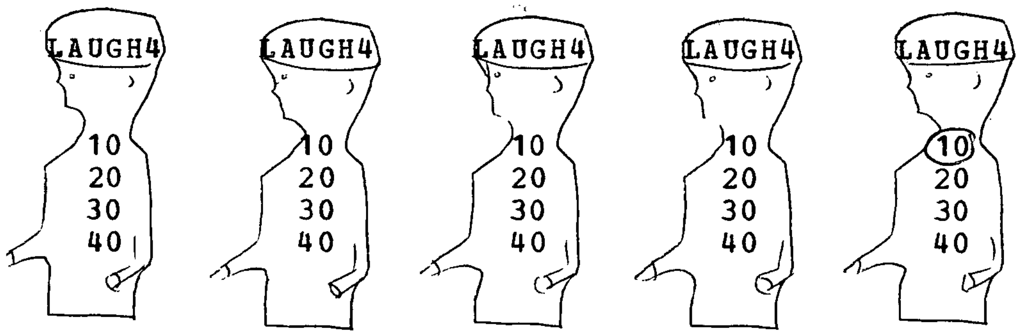
```
= LAUGH 4
   HA HA
   HA HA
   HA HA
   HA HA
```

With this new procedure LAUGH you have recursion and also the ability to stop it. Remember the ability to stop the procedure comes when the computer must make a decision. A computer can check to see if a number is zero, or whether a word is 'stop', 'go', or equal to another word. The decision to test these and other conditions yields an output of either True or False.

In the procedure LAUGH you are asking if :N: is equal to 0. If N does equal 0, you take the IFTRUE branch and stop. But if N does not equal 0, you take the IFFALSE branch and print out 'HA HA'. Then you subtract 1 from N and do the test again. N represents the number of times you ask the procedure to loop through the test. The example that follows shows a step by step trace through the procedure LAUGH 4.

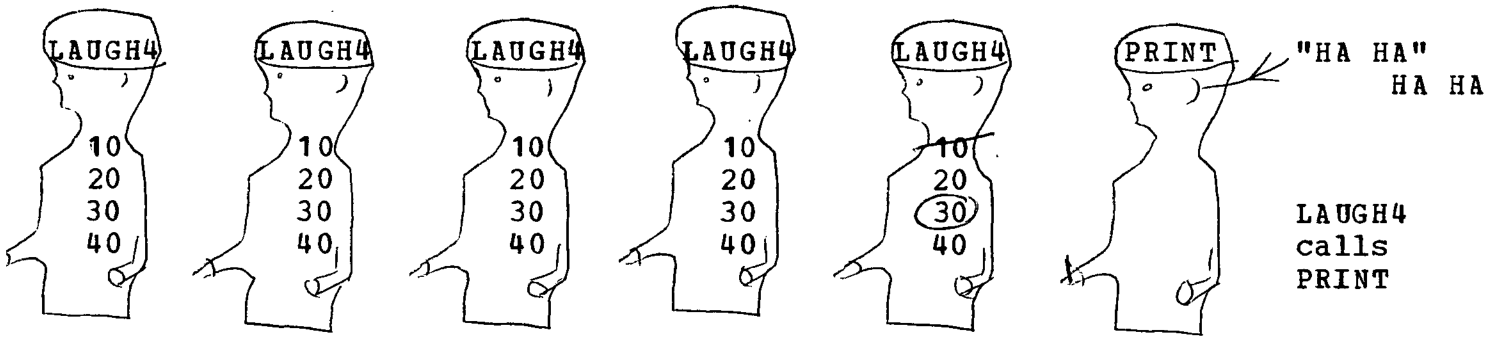


Snapshot 1



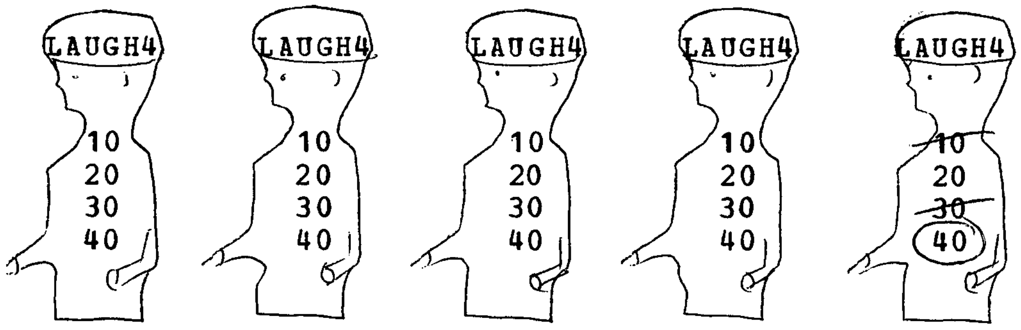
Snapshot 2

4 <> 0



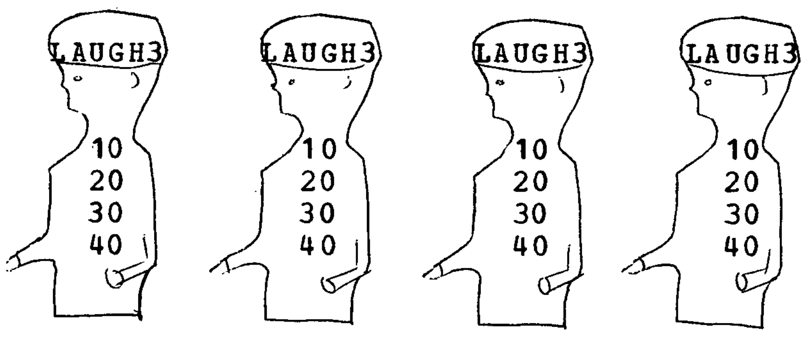
Snapshot 3

LAUGH4
calls
PRINT

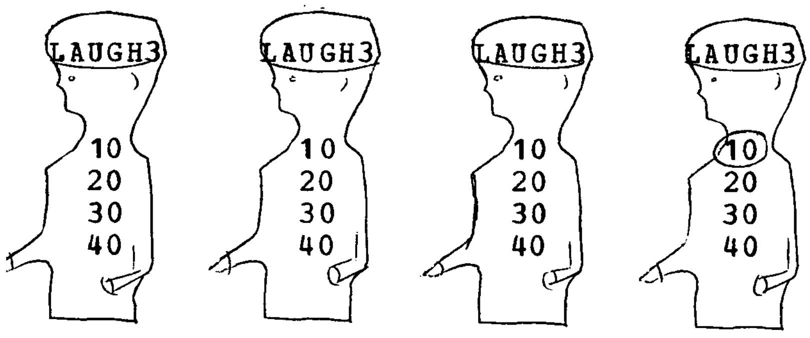


Snapshot 4

4 - 1 = 3
LAUGH4
calls
LAUGH3

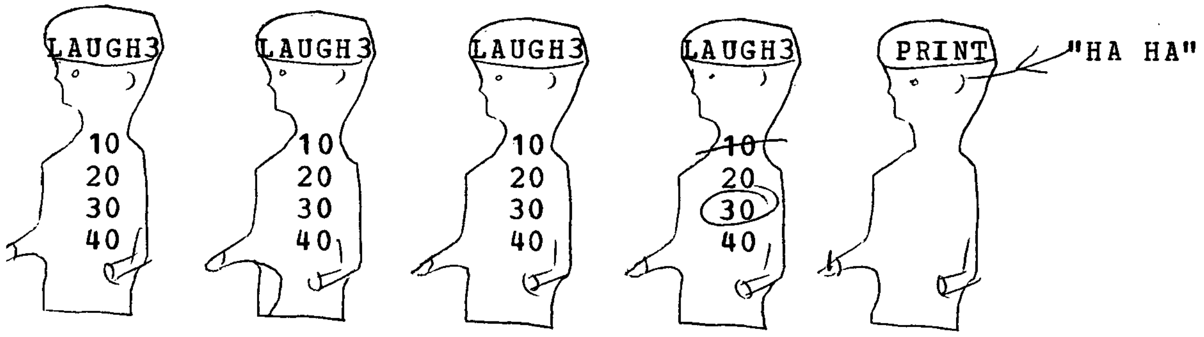


Snapshot 5



3 <> 0

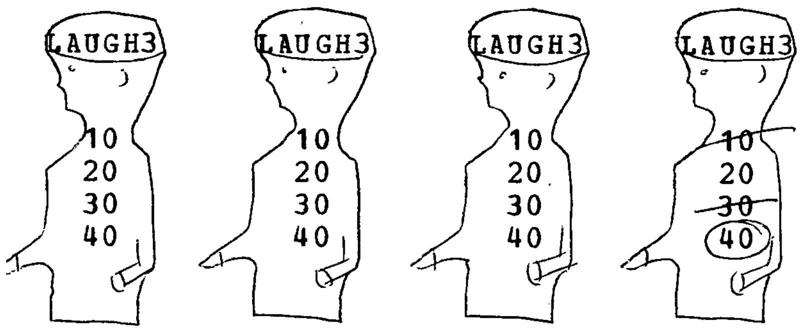
Snapshot 6



HA HA
HA HA

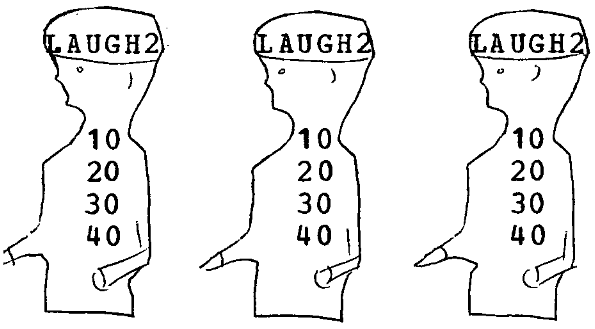
LAUGH3
calls
PRINT

Snapshot 7

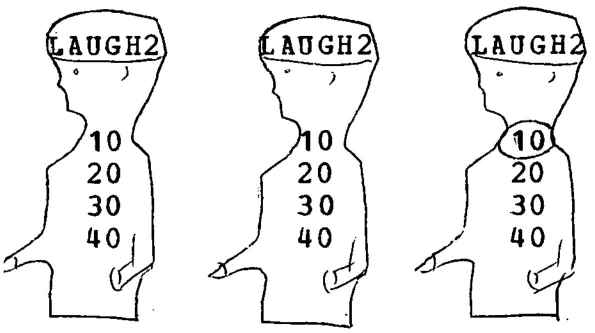


3 - 1 = 2
LAUGH3
calls
LAUGH2

Snapshot 8

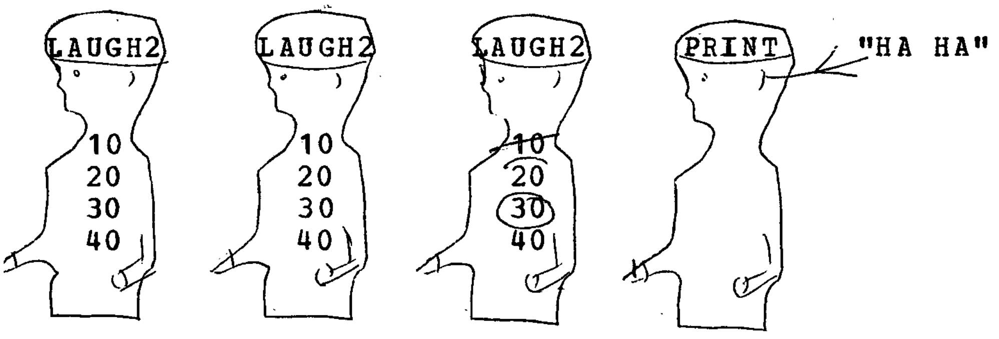


Snapshot 9



Snapshot 10

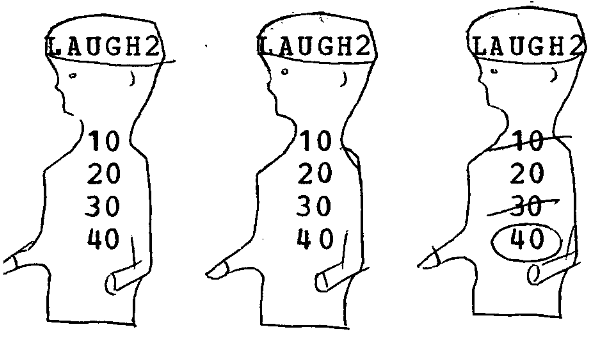
2 <> 0



Snapshot 11

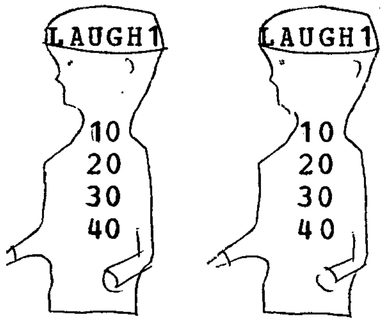
```
HA HA
HA HA
HA HA
```

LAUGH2
calls
PRINT

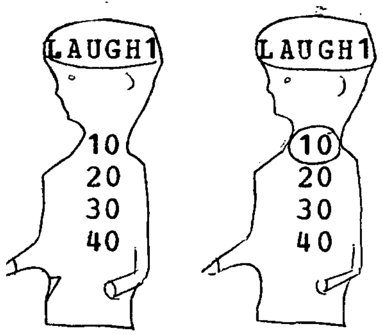


Snapshot 12

2 - 1 = 1
LAUGH2
calls
LAUGH1

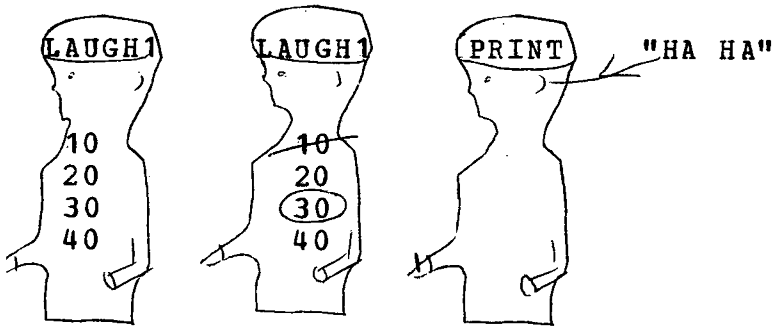


Snapshot 13



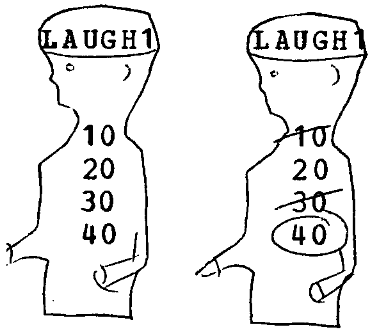
1 <> 0

Snapshot 14



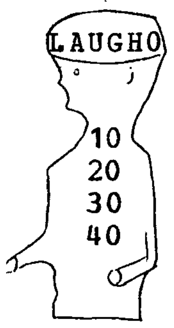
```
    HA HA
    HA HA
    HA HA
    HA HA
LAUGH1
calls
PRINT
```

Snapshot 15

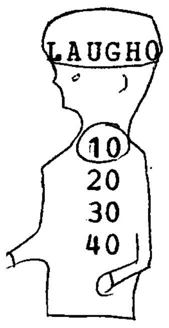


1 - 1 = 0
LAUGH1
calls
LAUGH0

Snapshot 16

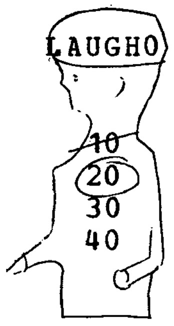


Snapshot 17



0 = 0

Snapshot 18



STOP

Snapshot 19

The only time you use quotes in an input procedure is when the input is a word. Look back to the procedure RESPONSE - page 68. As you see, the word 'KIMBERLY', is an input, and it is enclosed in quotes; the number 4, is also an input, and it is not enclosed in quotes. Why don't you try the procedure RESPONSE without quotes around the input word and see what happens. Do the same thing with the procedure LAUGH.

The command OUTPUT can only be used in a procedure; it returns control to the calling procedure, and outputs its arguments. By using OUTPUT the user can define procedures which are operations. See the glossary for example of using OUTPUT.

7. Editing a Procedure

Once you have defined a procedure, you may want to change something within it. To do this you need to call the command EDIT. For example:

```
= TO LAUGH :N:
@ 10 TEST IS :N: 0 (SEES IF THE VALUE OF N = 0)
@ 20 IFFURE STOP (IF TRUE THEN STOP)
@ 30 IFFALSE PRINT "HA HA" (IF FALSE PRINT HA HA)
@ 40 IFFALSE LAUGH DIFFERENCE :N: 1 (ALSO IF FALSE
          1 WILL BE SUBTRACTED FROM THE VALUE N)
@ END (THIS WILL END THE PROCEDURE)
LAUGH DEFINED.
```

If you had run LAUGH with the mistake on line 20, LOGO would return an error message to the screen as follows:

```
I DO NOT KNOW HOW TO IFFURE.
LOGO WAS EXECUTING LINE 20 OF LAUGH.
```

From this error message you are told of the misspelled word on line 20. To be able to correct the misspelled word you must first type in the command EDIT. EDIT by itself will not help you, so what you need is the word EDIT followed by the name of the procedure you wish to work on. For example:

```
= EDIT LAUGH (EDIT IS A COMMAND
        TO LOGO AND LAUGH IS THE PROCEDURE
        YOU WISH TO WORK ON)
```

You must be able to see what is in the procedure Laugh before you can make any changes. To do this the command SHOW is used. SHOW will print out everything in the procedure. For example:

```
= EDIT LAUGH
@ SHOW
  TO LAUGH :N:
  10 TEST IS :N: 0
  20 IFFURE STOP
  30 IFFALSE PRINT "HA HA"
  40 IFFALSE LAUGH DIFFERENCE :N: 1
  END
```

Now, with the procedure before you, you can correct the spelling error. You correct it simply by retyping line 20. LOGO will not let a procedure be run if there are two lines with the same number, so the most recent line is saved and the old line is forgotten. So to correct the procedure LAUGH, you type in the following line:

```
@ 20 IFTRUE STOP
```

The old line 20 with the misspelled word is forgotten and the new line 20 with the correct spelling is remembered.

```
= EDIT LAUGH
@ SHOW
  TO LAUGH :N:
  10 TEST IS :N: 0
  20 IFTRUE STOP
  30 IFFALSE PRINT "HA HA"
  40 IFFALSE LAUGH DIFFERENCE :N: 1
  END
```

END takes you out of editing mode while STOP terminates execution of the procedure and returns control to the calling procedure.

Sometimes it is necessary to delete lines from a procedure. To do this you would again EDIT the procedure and call the command SHOW. Now, to delete a line you type in the number of the line and enter it with nothing else on the line but the number. For example:

```
@ 40
```

This will delete line 40 from the procedure currently being edited.

If you are make a typing mistake while defining a procedure and it is before you have entered that line of code, you can correct the mistake by pressing down the CTRL key at the same time holding down the H key. Press the H key a sufficient number of times to bring the cursor under the last good character. Then you continue to type as if nothing had happened. For example:

```
@ 20 IFFURE STOP ---------- FTRUE STOP
```

This method is used on the silent 700 terminal. On the 3270 terminal there is a left arrow which you use to do back steps. And on the VUCOM there is also a left arrow.

8. Procedures for Your Use

If you wish to run some procedures in either arithmetic or spelling, you must type in the following statement:

```
= GET "?????"
```

The '?????' Should be replaced with the name of the procedure you wish to run. For example:

```
= GET "FARM"
```

'FARM' is the name of one of the procedures that deals with spelling

The following procedures are available to you on request:

```
MATHL -- Deals with less-than problems
MATHG -- Deals with greater-than problems
MATHE -- Deals with equal problems
MATHS -- Deals with addition problems
MATHD -- Deals with subtraction problems
MATHP -- Deals with multiplication problems
MAXMIN -- Deals with maximum and minimum problems

FARM -- A spelling routine.
         You are asked to spell a kind of animal
         animal and then spell the noise the
         animal makes.

SPELL -- A spelling routine.
         You are given a noise and you must
         spell the kind of animal
         that makes that noise.

TRUST, BUMP AND DREAM -- Spelling routines.
         You simply respond to some
         questions that are asked and
         the answers you give will
         find their way into a story.
```

LEARN -- A learning routine. It simply shows you how to create a story of your own.

To terminate the spelling routines or the arithmetic routines you must type in the word 'STOP' and you will find yourself back with LOGO waiting for your next command.

Please remember that if you wish to run the math routines you must type in some additional information. If you wish to run the procedure MATHL you must type in the following before you can run MATHL:

```
GET "MATHL"
GET "MATHG"
```

The same goes for the other math routines. MATHG must be accompanied by MATHS, MATHS must be accompanied by MATHD, MATHD must be accompanied by MATHP, MATHP must be accompanied by MAXMIN and MAXMIN can be by itself. All are typed as shown above.

Please also remember that the math routines only take integers (whole numbers). 0 1 2 3 4 ...

9. <u>Example of writing a program</u>

The following is a program which tests one's understanding of arithmetic multiplication. This program demonstrates the use of most of the basic words that you will need when you write programs of your own. Another way of writing sentences is with the letter 'S'. Using 'S' you can fit every thing on one line. If you spell out the word 'Sentence' there will not be enough room for the rest of the statement if the statement is a long one.

```
= TO MULTI
@ 5 PRINT " * * * * * * * * * * "
@ 10 PRINT S "WHAT IS" S RANDOM S "*" S RANDOM "?"
@ 15 MAKE "A" REQUEST (ASKS YOU FOR A NUMBER
          AND PUTS IT INTO THE MAILBOX CALLED 'A')
@ 20 TEST IS F:A: 0 (TEST TO SEE IF THE FIRST
          ELEMENT IN THE MAILBOX 'A' = 0)
@ 25 IFTRUE GO 100 (IF IT IS TRUE GO TO LINE 100)
@ 30 IFFALSE GO 5 (IF FALSE YOU GO TO LINE 5)
@100 PRINT " * * * * * * * * * * "
@102 PRINT " END OF EXERCISE -- BYE -- "
@ END (END OF THE PROCEDURE) MULTI DEFINED.
```

Now to see what it will look like when you run Multi.

```
= MULTI

* * * * * * * * * *
WHAT IS 7 * 9?
 63
* * * * * * * * * * *
WHAT IS 6 * 7?
 42
* * * * * * * * * * *
WHAT IS 5 * 6?
 30
* * * * * * * * * *
WHAT IS 9 * 1?
 0
* * * * * * * * * * *
END OF EXERCISE -- BYE --
```

Now you will see a modification of Multi.

```
= TO MULTI
@ 5 PRINT " * * * * * * * * * "
@ 10 PRINT "PLEASE TYPE IN TWO NUMBERS"
@ 15 MAKE "X" REQUEST MAKE "Y" REQUEST
          (ASKS FOR TWO NUMBERS)
@ 20 PRINT S "WHAT IS" S :X: S "*" S :Y: "?"
@ 25 MAKE "A" REQUEST (TYPE IN THE ANSWER)
@ 30 PRINT "NOW LET US SEE IF
             YOU ARE CORRECT."
@ 35 PRINT S :X: S "*" S :Y: "="
@ 40 PRINT PRODUCT F:X: F:Y:
          (PRINTS THE PRODUCT OF THE FIRST
          ELEMENT IN EACH OF THE
          MAILBOXES X AND Y)
@ 45 TEST IS F:A: PRODUCT F:X: F:Y:
          (SEES IF 'A' IS THE ANSWER)
@ 46 IFTRUE PRINT "GREAT YOUR ANSWER WAS
          CORRECT" GO 50 (JUMP TO LINE 50)
@ 47 IFFALSE PRINT "LET US TRY AGAIN.
@ 48 IFFALSE PRINT "GIVE ME THE SAME NUMBERS
          AS BEFORE." GO 50
@ 50 TEST IS F:X: 0
          (IF X=0 THEN YOU WILL QUITE)
@ 51 IFTRUE GO 100
@ 52 IFFALSE GO 5 (GOES BACK TO 5)
@ 100 PRINT " * * * * *"
@ 101 PRINT "END OF EXERCISE -- BYE --"
@ END (END OF THE PROCEDURE) MULTI DEFINED.
```

Now to see what it will look when you run Multi.

```
= MULTI
* * * * * * *
PLEASE TYPE IN TWO NUMBERS
4 5
WHAT IS 4 * 5 ?
20
4 * 5 = 20
WELL LET US SEE IF YOU ARE CORRECT
20
GREAT YOUR ANSWER WAS CORRECT
* * * * * * *
0 0
WHAT IS 0 * 0 ?
0
0 * 0 = 0
WELL LET US SEE IF YOU ARE CORRECT
0
GREAT YOUR ANSWER WAS CORRECT
* * * * * *
END OF EXERCISE - - BYE - -
```

10. LOGO Functions

The intent of this section is to show the user the various functions and a vague idea of what they do [MANIS, 1973].

10.1 Arithmetic

SUM, DIFFERENCE and PRODUCT do the familiar operations of addition, subtraction, and multiplication.
DIVISION outputs a sentence of the quotient and remainder from a division. QUOTIENT and REMINDER can be used to compute just one or the other result.
RANDOM outputs a random digit.
MAXIMUM and MINIMUM select the maximum or minimum of two numbers.

10.2 Commands

TO allows you to define a new function.
EDIT allows you to change the definition of an existing function.
END signals that you have finished defining or changing a function.
ERASE expunges the definition of a function.
ERASELINE allows you to delete a particular line of a function.
SHOW displays the definition of a function.
SHOWLINE displays a particular line of a function.
TITLE changes the title line of a function.
OUTPUT causes the currently executing function to return output's input as its value.
GOTOLINE transfers control within the currently executing function.
STOP terminates execution of the currently executing function.

10.3 Conditionals

TEST sets the truth flag from its input.
IFTRUE executes a statement only if the truth flag is true.
IFFALSE executes a statement only if the truth flag is false.

10.4 <u>Input - Output</u>

REQUEST reads in the a line of data, and outputs a <u>list</u> representing that line.
TYPE prints its input onto the user's console.
TYPEIN takes an input; it is equivalent to "MAKE input REQUEST".
PRINT behaves as TYPE does, but follows the printing by typing a carriage return.

10.5 <u>System Control Functions</u>

GOODBYE gets you out of LOGO and back to MTS.
GET allows you to add material from a file to the programs and variables with which you are currently working.
SAVE stores your current workspace in a file.
TRACE causes a specified function to be monitored for debugging purposes.
UNTRACE turns off debug monitoring for a function.
LOGOUT before you get out of LOGO, you will be asked if you wish a copy of your work, and in addition, this command will hang up the telephone line to the computer.

10.6 <u>Variables</u>

MAKE changes the value of a specified variable.

10.7 <u>Predicates</u>

IS is TRUE when its two inputs are equal.
WORDP is TRUE when its input is a word.
SENTENCEP is TRUE when its input is a sentence or list.
BOTH is TRUE when both of its inputs are.
EITHER is TRUE when either of its inputs is.
NOT is TRUE when its input is false.
NUMBERP is TRUE when its input is a numeric word.
GREATERP is TRUE when its first input is numerically greater than its second.
LESSP is TRUE when its first input is numerically less than its second.
EQUALP is TRUE when its inputs are numerically equal.
ZEROP is TRUE when its input is numerically equal to 0.

10.8 <u>Text Processing</u>

FIRST,BUTFIRST,BUTLAST and LAST allow you to take apart words, sentences or lists.
WORD, SENTENCE, and LIST allow you to create new things.

11. Glossary Of LOGO Functions

This section describes the set of LOGO functions currently available. The functions are listed in alphabetical order. If followed by a second version, this is the abbreviation which may be used if you do not want to spell out the entire word. Some LOGO words output values to other LOGO words. We call these words Operations. LOGO words which do not output are called Commands. These words command things to be done now. Words which switch to another mode or device are called Switches. Words which control the flow of a procedure (eg, Stopping, Going to a new line) are called Flow words [GOLDENBERG, 1975].

BOTH

(operation)
Outputs the logical of both its inputs which must be either TRUE or FALSE. (ie. TRUE if and only if both of its inputs are TRUE).

```
= MAKE "R" "TRUE" MAKE "S" "TRUE"
= MAKE "T" "FALSE"
= PRINT BOTH :R: :S:
TRUE
= PRINT BOTH :R: :T:
FALSE
```

BUTFIRST BF

(operation)
Butfirst outputs the list that contains all but the first element of the list.

```
= MAKE "A" "HELLO"
= PRINT BUTFIRST :A:
ELLO
= PRINT :A:
HELLO
```

BUTLAST BL

(operation)
Butlast outputs the list
that contains all but the last
element of the list.

```
= PRINT :A:
HELLO
= PRINT BUTLAST :A:
HELL
```

DIFFERENCE DIFF

(operation)
Given two numeric words, there
DIFFERENCE is outputed.

```
= MAKE "C" 6 MAKE "D" 5
= PRINT DIFFERENCE :D: :C:
-1
```

DIVISION

(operation)
Currently not working.

EDIT

(switch)
Edit is a command followed by a procedure
name. This command lets you add or
delete a line inside the procedure.
To switch off editing mode type END.

```
= TO LAUGH
@ 5 PRINT "HA HA"
@ 10 PRINT "BY"
@ END
LAUGH DEFINED.
= EDIT LAUGH
@ 10
@ SHOW
TO LAUGH
5 PRINT "HA HA"
END
@ END
LAUGH DEFINED.
```

EITHER

(operation)
Outputs TRUE if either of the
inputs are TRUE, FALSE if
both are FALSE.

```
= MAKE "S" "FALSE" MAKE "T" "FALSE"
= MAKE "R" "TRUE"
= PRINT EITHER :R: :S:
TRUE
= PRINT EITHER :S: :T:
FALSE
```

EMPTYP

(operation)
Outputs TRUE if the argument is
the empty word or the empty list,
FALSE otherwise.
Currently not working.

END

(switch)
Tells LOGO you are finished editing
or defining a procedure. Switches off
editing mode.

```
= TO FOO
@ 5 PRINT "HI THERE."
@ END
FOO DEFINED.
= EDIT FOO
@ 10 PRINT "THIS IS A TEST."
@ END
FOO DEFINED.
```

EQUALP

(operation)
Outputs TRUE if its inputs are
numerically equal, and FALSE
if they are not.

```
= MAKE "A" 6 MAKE "B" 5 MAKE "C" 6
= PRINT EQUALP :A: :B:
FALSE
= PRINT EQUALP :A: :C:
TRUE
```

ERASE

(command)
Erases the procedure from the workspace.
A procedure may not be erased while it
is being edited or defined.

```
= SHOW FOO
TO FOO
5 PRINT "HI THERE."
10 PRINT "THIS IS A TEST."
END
= ERASE "FOO"
= SHOW FOO
FOO IS NOT A PROCEDURE.
```

ERASELINE

(editing command)
Removes that line from the procedure
you are in. Currently not working.

FIRST F

(operation)
If the input is a list, First outputs the first element of the list, which may be a word or a list itself. If the is a word or a number, First outputs the first character of the word.

```
= MAKE "A" "HI THERE"
= MAKE "B" "HITHERE"
= PRINT FIRST :A:
HI
= PRINT FIRST :B:
H
= MAKE "C" 766 888 MAKE "D" "766 888"
= PRINT FIRST :C:
7
= PRINT FIRST :D:
766
= PRINT :D:
766 888
```

GET

(command)
Get followed by a procedure name. This procedure has been saved and put in to a storage file. Calling this procedure enables you to take the procedure from the file and work on it.

```
= GET "FOO"
FILE FOO SAVED ON YEAR MONTH DAY
```

GOODBYE BYE

(command)
Prints out a sweet message from LOGO and returns you to the operating system.

```
= GOODBYE
WOULD YOU LIKE A HARD COPY
OF THIS SESSION?
#
```

GREATERP

(operation)
Outputs TRUE if the first argument is numerically greater than the second, and FALSE if they are not.

```
= MAKE "C" 8 MAKE "D" 9
= PRINT GREATERP :C: :D:
FALSE
```

GOTOLINE GOTO GO

(flow)
Used in a procedure to transfer control to that line of the procedure.

```
= TO BOO
@ 5 PRINT "JUST A TEST USING GOTO'S."
@ 10 PRINT "TYPE IN A NUMBER"
@ 15 MAKE "A" REQUEST
@ 20 TEST IS FIRST :A: 0
@ 21 IFTRUE PRINT "GOODBYE" GO 100
@ 22 IFFALSE PRINT "A  = 0" GO 10
@ 100 PRINT "* * *" STOP
@ END
BOO DEFINED.
= BOO
JUST A TEST USING GOTO'S.
TYPE IN A NUMBER
9
A IS NOT ZERO
TYPE IN A NUMBER
0
GOODBYE
* * *
=
```

IFFALSE IFF

(control word)
Executes the line if the results of the most recent local Test was FALSE.

```
@ IFFALSE PRINT "A  = 0" GO 10
```

IFTRUE IFT

(control word)
Executes the line if the results of the most recent local Test was TRUE.

```
@ IFTRUE PRINT "GOODBYE" GO 100
```

IS

(control word)
Is returns TRUE if the two objects are the same and FALSE if they are not.

```
= MAKE "A" "HI THERE"
= MAKE "B" "HITHERE"
= MAKE "G" "HI THERE"
= PRINT IS :A: :B:
FALSE
= PRINT IS :A: :G:
TRUE
```

LAST L

(operation)
If the input is a list, Last outputs. The last element of the list, which may be a word or a list itself. If the input is a word or a number, Last outputs the last character of the word.

```
= MAKE "C" 789 MAKE "D" "789 999"
= MAKE "A" "HI THERE"
= MAKE "B" "HITHERE"
= PRINT LAST :A:
THERE
= PRINT LAST :B:
E
= PRINT LAST :C:
9
= PRINT LAST :D:
999
```

LESSP

(operation)
Outputs TRUE if the first argument is numerically less than the second. Otherwise FALSE.

```
= MAKE "C" 789 MAKE "D" "789 999"
= PRINT LESSP :C: :D:
789 999 IS NOT A NUMBER.
= MAKE "E" 789
= PRINT LESSP :C: :E:
FALSE
```

LIST

(operation)
Outputs a two-element list, whose FIRST is the first input and whose LAST is the second input. These inputs may be words, numbers or a list of things.

```
= MAKE "A" "HI"
= MAKE "B" "JUST A TEST"
= PRINT LIST :A: :B:
HI <JUST A TEST>
= PRINT LIST "HELLO," :B:
HELLO, <JUST A TEST>
```

LOGOUT

(command)
Control is returned to the operating system -- by signing you off with a message from LOGO.

```
= LOGOUT
```

MAKE

(command)
Assigns the name of the first element to have the value of the second element.

```
= MAKE "A" "HI THERE"
= PRINT :A:
HI THERE
= MAKE "F" :A:
= PRINT :F:
HI THERE
= MAKE "A" REQUEST
9
= PRINT :A:
9
```

MAXIMUM MAX

(operation)
Outputs the maximum of the two numeric arguments which are given.

```
= MAKE "C" 789 MAKE "D" 99
= PRINT MAXIMUM :C: :D:
789
```

MINIMUM MIN

(operation)
Outputs the minimum of the two numeric arguments which are given.

```
= PRINT MINIMUM :C: :D:
99
```

NOT

(operation)
Outputs TRUE if its input is FALSE and FALSE if its input is TRUE.

```
= MAKE "A" "HI THERE"
= MAKE "R" "TRUE" MAKE "S" "FALSE"
= PRINT NOT :A:
TRUE
= PRINT NOT :S:
TRUE
= PRINT NOT :R:
FALSE
```

NUMBERP

(operation)
Outputs TRUE if its input is a number and FALSE if it is not.

```
= MAKE "A" "HI THERE" MAKE "B" 666
= PRINT NUMBERP :A:
FALSE
= PRINT NUMBERP :B:
TRUE
```

OUTPUT

(flow operation)
Output tells the procedure to stop and output what it has in its hand.

```
= TO FOO
@ 5 PRINT "HI THERE"
@ END
FOO DEFINED.
= OUTPUT FOO
HI THERE
```

PRINT P

(command)
Print types its input on the console followed by a carriage return.

```
= PRINT "HI THERE"
HI THERE
= PRINT 789987
789987
```

PRODUCT PROD

(operation)
Outputs the multiplication of the two numeric numbers which are given.

```
= MAKE "G" 8 MAKE "H" 9
= PRINT PRODUCT :G: :H:
72
```

QUOTIENT

(operation)
When dividing two numeric numbers Quotient will give you the answer.

```
= PRINT QUOTIENT :G: :H:
0
= PRINT QUOTIENT :H: :G:
1
```

RANDOM

(operation)
Outputs a one-digit random integer. Zero can be a possible output.

```
= PRINT RANDOM
9
= PRINT RANDOM
7
```

REMAINDER

(operation)
Outputs the remainder of two numeric numbers.

```
= PRINT REMAINDER :G: :H:
0
= PRINT REMAINDER :H: :G:
1
```

REQUEST

(operation)
Outputs a list.

```
= MAKE "A" REQUEST
9
= PRINT :A:
9
= PRINT :B:
= MAKE "B" :A:
= PRINT :B:
9
```

SAVE

(operation)
The procedure will be saved and put into a storage file, to be used at a later date.

```
= SAVE "FOO"
```

SENTENCE S

(operation)
Outputs a sentence made from the first input followed by the second.

```
= PRINT SENTENCE "HI" "THERE"
HI THERE
= MAKE "A" "HELLO"
= PRINT SENTENCE :A: "THIS IS A TEST"
HELLO THIS IS A TEST
```

SENTENCEP

(operation)
Outputs TRUE if its input is a sentence and FALSE if it is not.

```
= Make "a" "this is a test"
= make "c" 99
= print sentencep :a:
true
= print sentencep :c:
false
```

SHOW

(switch)
Prints out the definition of the specified procedure.

```
= SHOW FOO
TO FOO
5 PRINT "HI THERE"
END
= EDIT FOO
@ SHOW
TO FOO
5 PRINT "HI THERE"
END
@ END
FOO DEFINED.
```

SHOWLINE

(show command)
Currently not working.

STOP

(flow)
Stop causes the currently executing procedure to return.

```
= TO NUM
@ 5 MAKE "A" REQUEST
@ 10 TEST IS FIRST:A: 0
@ 11 IFTRUE PRINT "A IS ZERO" STOP
@ 12 IFFALSE PRINT "A  = 0" GO 5
@ END
NUM DEFINED.
= NUM
9
A IS NOT ZERO
0
A IS ZERO
```

SUM

(operation)
Outputs the sum of two numerical numbers.

```
= MAKE "A" 8 MAKE "B" 9
= PRINT SUM :A: :B:
17
= PRINT SUM 8 9
17
```

TEST (command)
Test runs its input, which must evaluate to TRUE or FALSE and saves the result until Iftrue or Iffalse can use the information.

```
= TO NUM
@ 5 MAKE "A" REQUEST
@ 10 TEST IS FIRST :A: 0
@ 11 IFTRUE PRINT "A IS ZERO" STOP
@ 12 IFFALSE PRINT "A  = 0" GO 5
@ END
```

TITLE (command)
Currently not working.

TO (switch)
Used to define procedures. The input Title must contain a name not already used by LOGO and may contain additional words prefixed by ":" which name inputs to the new procedure. To turns on the editing mode and is terminated by the switch End.

```
= TO FOO
@ 5 PRINT "HI THERE"
@ END
FOO DEFINED.
= TO NUM :N:
```

TRACE (switch)
Trace causes the specified procedure to print out every line while it is executing. Untrace stops the tracing action.

```
= TRACE FOO
HI THERE
= SHOW FOO
TO FOO
5 PRINT "HI THERE"
END
=
```

TYPE (command)
Like Print except it does not carriage return at the end of the line. Currently not working.

```
= TYPE FOO
HI THERE
```

TYPEIN

(operation)
Outputs only the first word as a word, similar to Request.
Currently not working.

UNTRACE

(switch)
Untraces the procedure that was traced.

```
= UNTRACE FOO
HI THERE
```

WORD W

(operation)
Outputs a word made from the chatacters from the first input followed by those from the second input. An input may be a number instead of a word.

```
= MAKE "A" "HI" MAKE "B" "THERE"
= PRINT WORD :A: :B:
HITHERE
```

WORDP

(operation)
Outputs TRUE if its argument is a word and FALSE otherwise.

```
= MAKE "A" "HI" MAKE "C" 9
= PRINT WORDP :C:
TRUE
= PRINT WORDP :A:
TRUE
= PRINT WORDP :D:
FALSE
= PRINT :D:
```

ZEROP

(operation)
outputs TRUE if its input is numerically equal to 0 and FALSE otherwise.

```
= MAKE "A" 8 MAKE "B" 0
= PRINT ZEROP :A:
FALSE
= PRINT ZEROP :B:
TRUE
```

12. Warning and error messages

Source errors are printed by the compiler as they occur after translation. LOGO explains what it is objecting to. The following are some examples:

```
= 12PRINT "HI THERE"
LOGO DOESN'T KNOW HOW TO 12PRINT

= PRINT S :RANDOM S "*" S :RANDOM: S "=" F:A:
YOU CAN'T USE RANDOM SENTENCE " "*" SENTENCE AS A NAME.
YOU CAN'T USE SENTENCE "=" F AS A NAME.
MISSING :

= TO MULTI
@ 14 PRINT PRODUCT F:RANDOM: F:RANDOM:
@ END

= MULTI
FIRST,BUTFIRST,LAST,AND BUTLAST DO NOT APPLY TO "".
LOGO WAS EXECUTING LINE 14 OF MULTI.


= PRINT HOW ARE YOU TODAY"
MISSING "
```

13. References

[1] Printed from the AISB European Newsletter (July, 1976), page 46.

Goldenberg, Paul E., A glossary of PDP11 LOGO primitives. MIT Artificial Intelligence Laboratory, Cambridge, Mass., AI Memo No. 315A (March, 1975).

Manis, Vincent S., A Machine Independent Implementation of LOGO. University of British Columbia, Department of Computer Science, M.Sc. Thesis (1973).

INDEX