Design and implementation of a generalised computer aided learning system

Nga, Me Hin

Design and Implementation of a Generalised Computer Aided Learning System.

Me Hin Nga

A Thesis submitted for the Degree of

Master of Science

at the University of Durham

Department of Computer Science
University of Durham

31 October 1986

Design and Implementation of a Generalised Computer Aided Learning System.

Me Hin Nga

Department of Computer Science
University of Durham

ABSTRACT

This thesis surveys the development of computer aided learning and outlines the tools that are used for creating computer aided learning systems. A project to create and port over a computer aided learning system from a VAX 11/750 to a PDP model 11/44 based on the UNIX operating system is described. The computer aided learning system makes extensive use of existing software tools available on UNIX and is hence named **CALUNIX** for Computer Aided Learning on **UNIX**.

Acknowledgements

I would like to thank the staff and colleagues at Durham who have given their help in this work, especially A.J. Slade, P.L. Davies and M. Storey.

My parents and family who have given me this opportunity and their continued support.

Finally, I am grateful and indebted to my supervisor, Mr. J.S. Roper, for his guidance and support in my work.

CONTENTS

CHAPTER 1

COMPUTER AIDED LEARNING

CHAPTER 1 COMPUTER AIDED LEARNING

1.1 Introduction

The Computer has greatly influenced the way society functions. What started as a research tool for scientists has extended its application to become tools in the information, commercial, entertainment, production, defence, education and many other areas. In this chapter the use of the computer as a learning tool and some of the educational theories and assumptions are examined.

1.2 History of Computer Aided Learning/Instruction

What is learning? Learning can be described as a continual process in which one is engaged in order to acquire the necessary knowledge and skills to survive and live within an environment. In the context of this thesis learning is taken as an activity in which the student is engaged in order for him or her to assimilate the relevant information and apply his understanding to achieve certain objectives. The objectives might be to solve certain problems posed or to be able to programme in a new computer language.

The computer is an electronic machine that is able to perform high speed operations and handle huge amounts of information. The operations the computer performs depend on the instructions given to it. The instructions which can be expressed in the form of code which the computer can interpret are known collectively as software.

Computer Aided Learning or Computer Aided Instruction are two common acronyms used, the latter in the United States of America and the former in Great Britain. Computer Aided Learning (CAL for short) is the process by which textual and graphic information is presented in some logical sequence to a student by a computer. Here the computer serve as an audio/visual/tactile device. The student learns by reading from the text material presented or graphical information displayed and interacting with the computer via the keyboard or other input devices. In this manner the computer can be developed into an effective instructional medium.

Instructional devices like the blackboard, overhead projector, etc. could be classified as instructor-centered. The change in the use of teaching devices to be learner-centered instead of instructor-centered could be seen as the beginnings of the Computer Aided Learning era. In the following sections we will trace the milestones Computer Aided learning has traversed up to the state it is at present.

1.2.1 Programmed Learning 1950's and Learning Theories.

One of the most important factors which contributed to the development of Computer Aided Learning was the growth of the programmed instruction or programmed learning movement.

Programmed learning or programmed instruction is a process whereby a learner is presented with a small sections of material to be learnt and to which some form of response is required. The required response is put in such a way that it should almost always produce a correct one by the learner. This stimulus and response method is geared to direct the learner towards a desired behaviour.

The programmed learning technique originated mainly from the work done by Sidney Pressey and B.F. Skinner between 1920 and 1955. In 1924 Pressey used a machine for grading multiple choice examinations. The idea was furthered by Skinner in the experiments carried out on learning behaviour.

Skinnerian (Behaviourist) Learning model.

Skinner's model of behaviour known more commonly as 'Operant Conditioning' can be described in terms of stimulus and response. Learning is deemed to have occurred when a specific response is elicited by a specific situation or stimulus with a high degree of probability. The more predictable the response, the more efficient the learning is deemed to have been. The learning of more complex behaviour can be described as building up a chain of stimulus and response bonds.

The programmmed learning technique was described in an experiment on pigeons being trained to perform a specific task. The pigeon is rewarded each time it does a correct response. The desired behaviour is thus shaped and reinforced by rewarding the correct actions. This principle is extended to shaping the behaviour of the learner by presenting a gradual progression of small units of information and related tasks. Each task being created to ensure maximum likelihood of success.

Papert's Communication Model of Learning.

Another approach that influenced the scope of programmed learning at about the same time as B.F. Skinner was when Seymour Papert advocated a 'skill analysis approach'. The approach is applied as a step by step procedure such as the solution of a geometry problem or the assembly of a mechanism that can be readily described as a series of steps in a chain and then adding the next one. The skill analysis approach towards learning is characterised by

(1) A thorough analysis of the task to be taught.
(2) List knowledge and skills requirements separately.
(3) Use observation and interview methods to extract the requirements.

Seymour Papert gave no indication of being influenced by Skinner's work and he has based the approach on the communication theory of learning as opposed to Skinner's reliance on the behaviourist theory of learning. (See Figure 1.0 - Behaviourist learning model, Figure 1.1 - Communication model). Though the approaches taken by Skinner and Papert are different the resulting course structures developed when Skinner's approach was applied to skills training is almost identical.

Problems with the programmed instruction technique arose when the disciples of the proponents, Skinner and Papert, tried to extend the domain of application to all forms of learning.

Gange's Hierarchy of Learning

Robert Gange has classified the basic learning concepts in a way which represents a hierachical view of how the concepts of learning are acquired. (See figure 1.2 Gange's Hierarchy of Learning Categories.) The hierarchy was meant to show that learning proceeds from a simple conditioning type to more complex problem solving type. Implicit in the hierarchy is also the precondition that lower levels of learning should be attained before higher levels could be tried. A brief description of each level of learning follows:-

(a) Signal Learning.

This is equivalent to the Pavlovian conditioned response type of learning. The subject here learns that a given event is the signal for another. The illustration by Pavlov is conditioning the dog (subject) to salivate (response) upon a dinner bell being rung (signal for another event) instead of the actual dog's dinner (event). The critical factor is the timing of the events, the bell being rung after the dog has been given his dinner or if the time between the bell signal and the dinner is too far apart would not elicit the required response at the bell signal event.

(b) S-R Learning (Stimulus-Response Learning).

This is different from signal learning in that the response is not a generalised emotional one, but a very precise act. Gange gives the illustration of a dog learning to respond to its master's command of 'shake hands' by offering its paw. The characteristics of S-R learning are:-

(1) The learning is typically gradual with some repetition of the association between the stimulus and response being necessary.

(2) The response becomes more sure and precise as the repetition progresses.

(3) The controlling stimulus becomes more precise - initially the dog may respond to a variation of the commands but as

the training progresses these variations cease to produce response.

(4) Some form of reward (or reinforcement) is given, when a correct response is produced.

(c) Chaining

A chain is a series of simple stages that go to make up a procedure. It may be represented by a chain of S-R bonds and may be implemented either by Skinner's operant conditioning method or by Papert's progressive parts method. Characteristics of chaining are:-

(1) Each link(connector) in the chain must be established first.

(2) Time is a factor, events need occur close in time.

(3) If the preceding two conditions are satisfied, learning a chain occurs on a single occasion. However the occasion might need to be repeated if the links are not well established.

Verbal chaining is a sub-variety of chaining. Gange gives the example of learning French for match (allumette) in the following way. The word 'match' can be associated with the mental picture of a match illuminating. So a chain is established.

(d) Discriminant Learning

Discriminant learning requires that the necessary S-R effects are already established and the interference from conflicting stimuli must be reduced to a minimum.

(e) Concept Learning

Concept learning classifies a stimulus in terms of its abstract properties. An example is given of a child learning that a blue block X is called a cube and another block Y twice the size of X is red in colour, is also a cube. Concept learning would enable the child to identify a cube on the basis of an internalised representation (shape) which is independent of the dissimilarities (colour) of the two objects. The characteristics are:-

(1) The initial S-R portions of the chains must be learnt.
(2) A variety of stimulus situations must be presented, so that the conceptual property common to all of them can be discriminated.
(3) The learning of a new concept may be gradual, because of a need to introduce a variety of stimulus situations.

(f) Rule learning

In the formal sense, a rule is a chain of two or more concepts. The simplest type of rule may be 'If A then B'. Once a rule is correctly learnt the learner will be able to apply it in all relevant situations, but he may not be able to state the rule in words. The characteristics are:-

(1) The concepts to be linked must be clearly established - the learner must know what a 'feminine' noun is and what a 'feminine article' is.
(2) A simple process of chaining can take place.
(3) The learning of a rule may take place in a single occasion.

(g) Problem solving

Once some rules are acquired a person can combine these rules into a great variety of higher order rules. In doing this he can apply what he already knows to solve problems which are new to him. The characteristics are:-

(1) The learner must be able to recognise the features of the response that constitute the solution to the problem.
(2) Relevant rules are recalled and used.
(3) The recalled rules are combined so that a new rule emerges.
(4) Though the process of arriving at a solution may take a very long time Gange thinks that the actual solution is actually arrived at a 'flash of insight'.

The learning concepts presented here form the basis of some of the early and later CAL systems that are described in the next section.

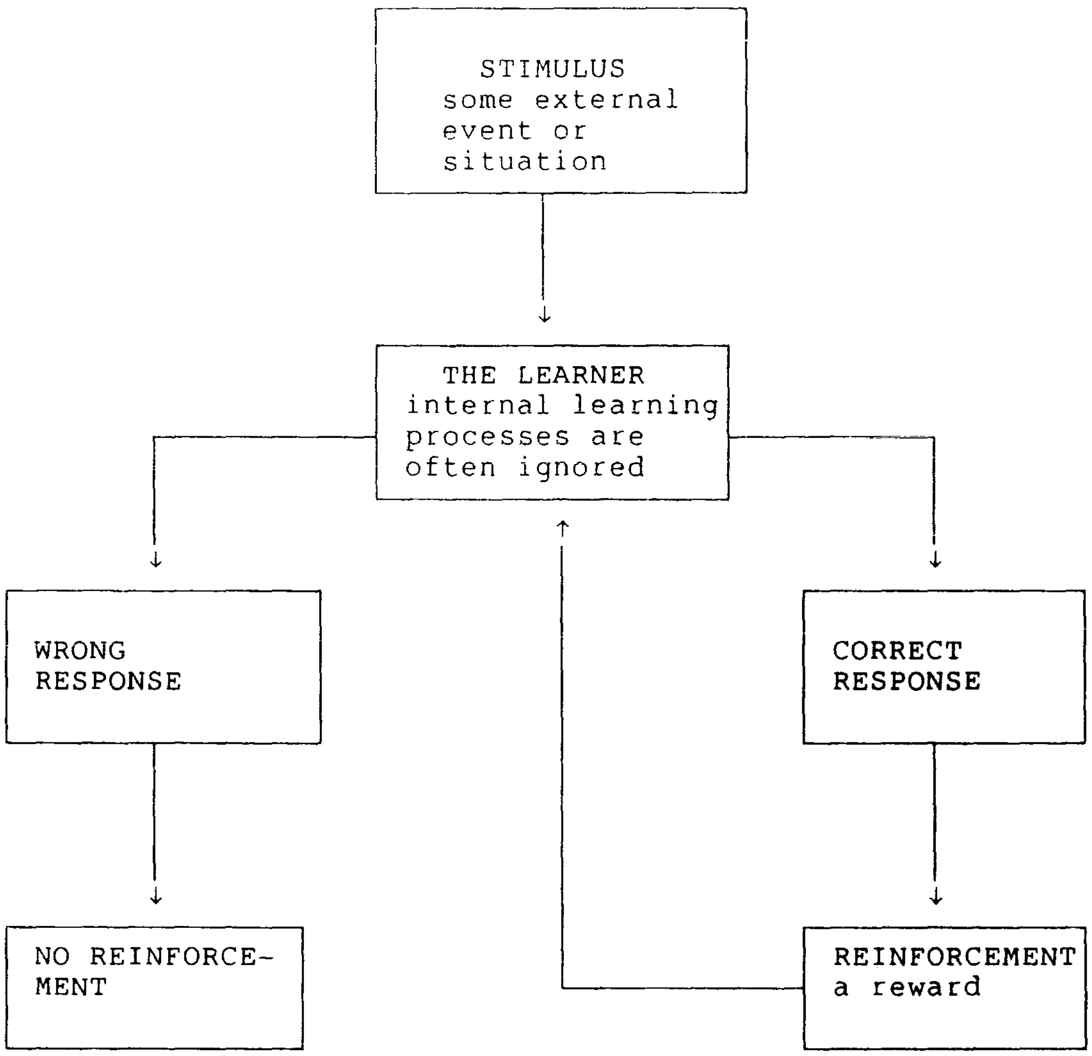


Figure 1.0 A Behaviourist Learning Model

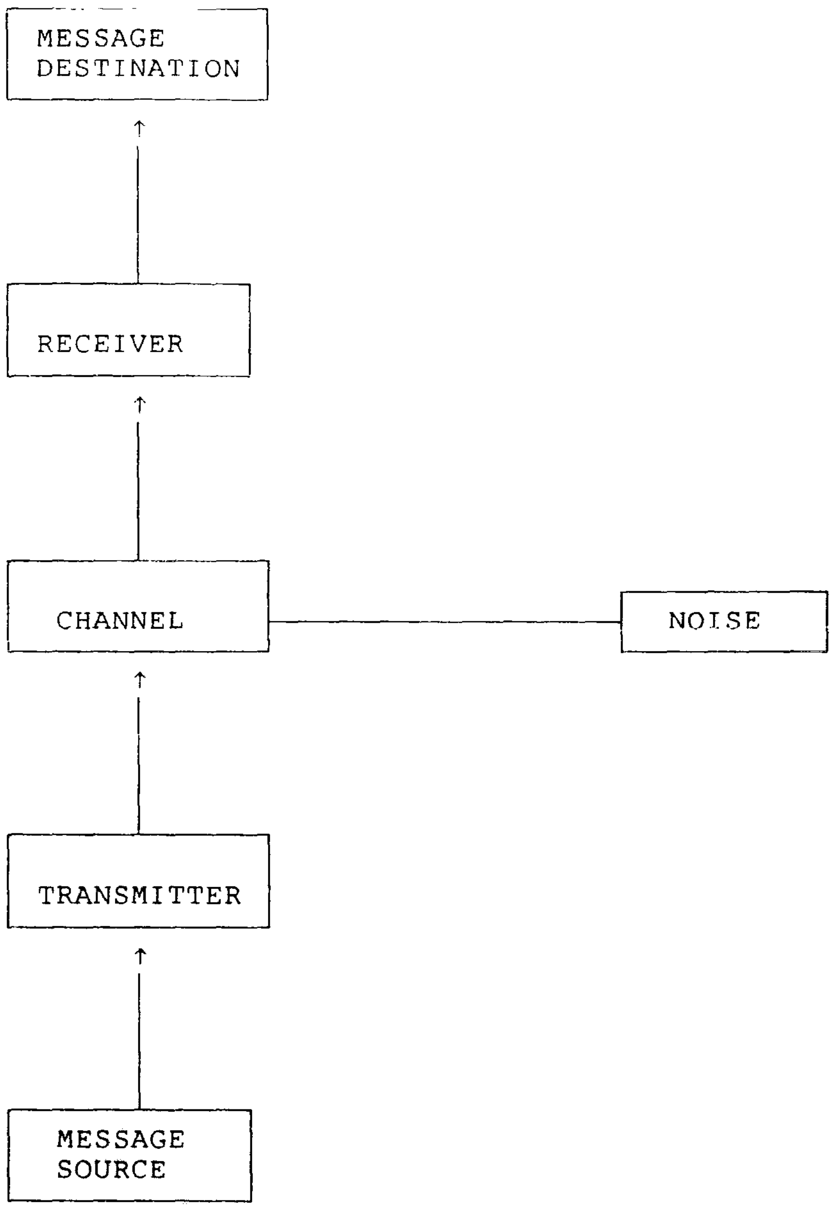


Figure 1.1 An engineer's model of the communication process (From C.E. Shannon and W. Weaver, The Mathematical Theory of Communication, University of Illinois Press, 1949)

Any conflicting messages received at the destination are regarded as 'NOISE'. Efficient communication aims at reducing this 'NOISE'.

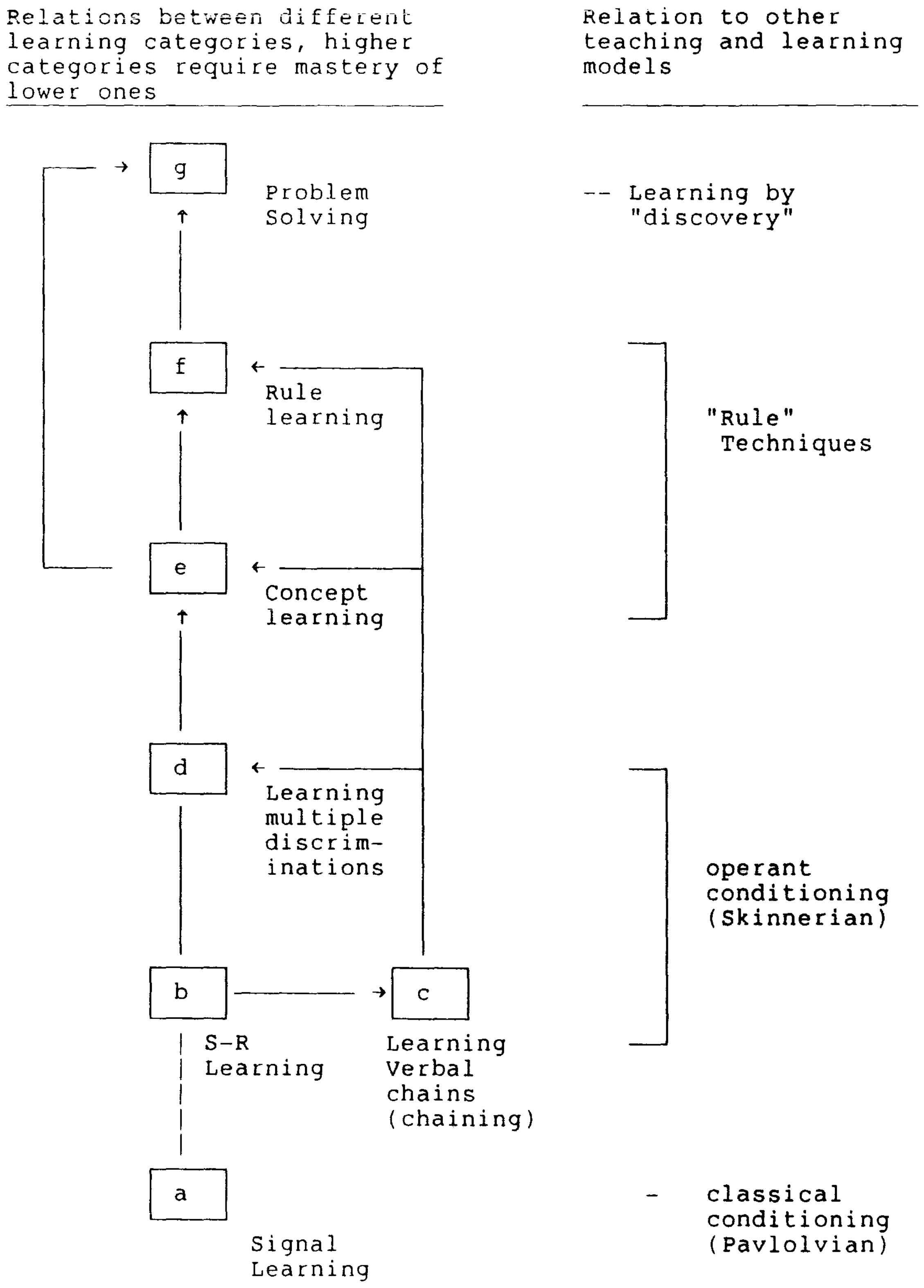

Relations between different learning categories, higher categories require mastery of lower ones
Relation to other teaching and learning models
g
Problem Solving
-- Learning by "discovery"
f
Rule learning
"Rule" Techniques
e
Concept learning
d
Learning multiple discrim-inations
operant conditioning (Skinnerian)
b
c
S-R Learning
Learning Verbal chains (chaining)
a
Signal Learning
- classical conditioning (Pavlolvian)

Figure 1.2 Gange's Hierarchy of Learning categories

1.2.2 Computer Aided Learning Systems

Early successes in the use of computers in aiding scientific research and administration have led to the appraisal of its application in other areas. During the 1960's the computer was beginning to be used as a tool in the educative process. The use of computers for direct instruction has proved to be a challenging, long and difficult task. This section surveys some of the past work in Computer Aided Learning. The presentation is not in a strict chronological order, but aims to show the development of differing types of ideas and approches.[Atkins69,Bitzer70] Table 1.0 shows some of the approaches to Computer Aided Learning.

In the following section we will survey some CAL approaches in some detail.[Self83]

APPROACH	CHARACTERISTICS/ILLUSTRATION
Linear programmes	Derivation from behaviourism; Systematic preparation; Reinforcement and self-pacing. Drill and Practice.
Branching programmes	Corrective feedback; adaptive to Student response; tutorial dialogues; use of author languages.
PLATO	Multi-terminal interactive system; visual displays; 'open shop' approach; expensive.
TICCIT	Team production of courseware; 'mainline' lessons; use of television and minicomputers; learner controlled.
Generative CAI	Ease burden of preparation of teaching material; precursor intelligent tutoring systems; Drill and Practice.
Expert Systems	SCHOLAR; SOPHIE; GUIDON SPIRIT.
Games	Intrinsically motivating; audiovisual effects; often lacking educational aims; WEST.

Table 1.0 Approaches to CAL

Drill and Practice

Drill and Practice systems present practice problems and exercises to reinforce learning gained from another source. In addition to keeping track of right and wrong answers, the computer can provide useful student feedback and remedial information. One of the early CAL systems developed in this mould was at Stanford University around 1967. The Drill and Practice system was based on a Digital Equipment Corporation PDP1 central processing unit with a high speed drum and a model 33 teletype unit as the student interface. The Drill and Practice system updates off-line the material to be presented to the student the next day. This material is dependent upon the performance recorded on that day.

Tutorial System

The Tutorial approach is essentially programmed instruction implemented on a computer. The computer presents the material to be learned in sequential frames. Either linear or branching modes of programmed instruction can be used. The Tutorial system was also developed in Stanford University in parallel with the Drill and Practice System. Initially the system used a Digital Corporation Equipment PDP1 central processing unit with an attached IBM 7090 disc drive. The system was later developed on another system(IBM 1500) and an author language called COURSEWRITER II was used.

The 1500 system consists of an IBM1800 central processing unit, tape unit and exchangeable discs. Student terminal consisted of a cathode ray tube, typewriter keyboard and a light pen. This system has been classified as a tutorial system due to a branching structure which allows for real time instructional decisions to be made on what material is to be presented next based on the student's last response or upon an evaluation of some subset of his total response history.

The Tutorial and the Drill and Practice procedures described in the context of the Stanford University' projects are by far the most prevalent modes of computer aided learning strategies. Their value lies in the individualised nature of interaction between computer and student which is described as an optimising process.

This optimising process starts with a lengthy preparation of the learning material for a short interaction, in the course of this, involving preliminary testing on students and removal of fuzzy or dull or otherwise inadequate portions of the material. In the end an incisive educational piece of material is used.

The computer can keep a record of the interaction with the student. For the educator this record of performance allows a means of studying students' conception and misunderstandings or the inadequacies of the learning material. For the student, a record of his particular learning idiosyncrasies can govern the heuristics or rules of the thumb used by the his "tutor" (the CAL system). The CAL system can respond more actively or draw out the student whether it tells or asks, whether ideas are best presented first by example or introduced at once as general principles, whether small steps and repetitions or great mental strides are needed, whether visual or auditory presentation is most helpful and so on. The third gain of the performance record is that, at the end of a block, the student as in fact demonstrated mastery and has passed his examination, the computer thus can 'teach' and certify achievement.

The opportunity is offered to the student to learn at his own convenience of time, place, pace.

PLATO

The largest Computer Aided Instruction exercise undoubtedly belong to the PLATO project in the United States of America. The PLATO system originated at the University of Illinois by Bitzer and Braunfeld during the early 1960's. The PLATO project was funded by the National Science Foundation of America and later Control Data Corporation took over the project on a commercial basis.

The original system had a terminal that supported both alphanumeric/graphical display as well as the facility to superimpose computer selected slides onto the terminal display. The subjects taught cover mathematics, language drills and computer related topics.[Bitzer76]

The first PLATO system consists of one terminal link to an ILLIAC I computer system and fifteen years later(1975) the configuration supports over 900 terminals linked to a huge main-frame, a CDC CYBER 73-2. The scale of PLATO use grew to more than 1 million student contact hours in the year 1975 with over 4000 lessons representing approximately 3500 hours of instructional material in over 100 subject areas. If success is measured in terms of how much a system is being used then PLATO would certainly be on top of the list of the most used CAL systems then and even now. Critics of PLATO and other CAL systems often used cost effectiveness as a main measure of success and PLATO have not been deemed to be successful in this respect.[Yeates81]

The main features of the PLATO system are:-

(1) A main-frame based system capable of supporting over 900 terminals.

(2) Specialised terminals using a gas matrix display panel capable of addressing 512 by 512 positions on the screen. This display panel being constructed of flat transparent glass permits slide images to be projected from the rear on the graphic display. The images are stored on a microfiche image sheet and any one of the 256 images can be selected within 0.2

second. Audio facility on the terminal allows as many as 4096 messages or 22 minutes of audio. The messages are stored on a disc device that can fetch a message within 0.3 second. The terminal also permit authors to generate their own set of 126 characters in addition to the set of 126 always available in the terminal. Besides the keyboard, users can enter information via a touch panel. The touch panel consists of a transparent plastic film containing light-emitting diodes and diode detectors on a 16 by 16 matrix mounted on the front display panel. Finally each terminal has an additional input-output connector for attaching any other devices.

(3) Author language TUTOR allowed classroom teachers to write the PLATO lesson materials.

(4) A mix of teaching strategies is often used in teaching most of the PLATO subjects. Strategies include drill and practice, simulation, tutorial, and dialogue.

(5) Two types of hardcopy devices are supported:- a video hardcopy unit reproduces the video image from the screen onto paper and an alphanumeric printer.

Because of the specialised equipment and software needed to support the PLATO system, cost has always been a worry. In the United Kingdom only large institutions like the Royal

Navy, International Computers Limited and International Telephone and Telegraph, initially have been able to use the PLATO system. Evaluation reports on the empirical data collected during the five year PLATO project gave no compelling evidence that PLATO had a positive or negative impact on student achievement or dropout rate. The subjective feedback in the form of questionnaires gave evidence of PLATO students' favourable attitudes towards computers and computer assisted instruction compared with non-PLATO students'. Teachers' reaction towards PLATO was also favourable, the authors of the evaluation speculated that the high acceptance by the teacher to the fact that teachers perceived that they retained control over how PLATO was used, and that the system therefore was not a threat to their current procedures. Cost of PLATO was three times more than the targeted figure that otherwise would have made PLATO an equal cost alternative to more traditional teaching methods. In all PLATO made the very significant impact in the world of CAL and its ideas and methods are still retained in many other CAL systems.

TICCIT

TICCIT stands for Time-Shared Interactive Computer Controlled Information Television. At about the same time as the PLATO system another project known as TICCIT was started. Both PLATO and TICCIT were funded under a 10 million U.S.

dollar project spreading over five years by the National Science Foundation of America. TICCIT project was directed by the Mitre Corporation whom also were responsible for the hardware and software for a computer assisted learning delivery system. Another group led by Victor Bunderson at the Brigham Young University developed the courseware for the system.

The aim of TICCIT project was to create a marketable system that could be used as a main media for delivery of instruction. This approach of selling a mainline computer assisted instruction package has resulted in the use of "off-the-shelf" products partly to minimise the costs, increased the reliability of the tested components and wider user acceptance.

The hardware used in TICCIT is made up of two General Nova 800 series minicomputers with up to 128 terminals connected to the system. Each of the terminals had a Sony colour television set to display alphanumeric and graphic characters in seven colours. Input consists of an alphanumeric keyboard with a special set of "learner control" keys and an optional light pen that permits a user to point and receive input from a specific location on the television screen.

TICCIT's method of instruction put emphasis on the learner being in control of the course material. This is facilitated by means of a high level command language incorporated in the special set of "learner control" keys shown in figure 1.3. The lower nine set of keys are used by the learner to control his own learning tactics. For example the **OBJ'TIVE** key will access an illustration of the segment objective, the **MAP** key accesses the next higher level for status or survey, and the **ADVICE** key elicits the adviser programme comments and strategy. The **HELP, HARD** and **EASY** keys are used in conjunction with the **RULE, EXAMPLE** and **PRACTICE** keys. The **RULE, EXAMPLE** and **PRACTICE** keys may be accessed in any order and may also be repeated. When a student feels he has mastered the material requested, he may ask for a "TEST". The computer grades the test and informs the student of the results. If the student fails the mastery test, advice is given as to which material he must review before he tries another test.

The National Science Foundation gave a draft evaluation report on TICCIT in 1977. Some of the report's main conclusions were:-

(1) TICCIT exerted a significant impact on student achievement in both mathematics and english composition. Students who completed courses under a TICCIT programme generally attained higher scores than similar students in a lecture discussion environment.

(2) Low completion rates for TICCIT courses compared

with non-TICCIT courses a were worrying result. Some explanation for this phenomenon were attributed to TICCIT's bias towards high ability students to the detriment of the less able ones and also insufficient degree of instructor involvement in managing the students' progress.

(3) The evaluation concluded that TICCIT had confirmed on the potential of computer aided instruction as an effective resource in student learning.

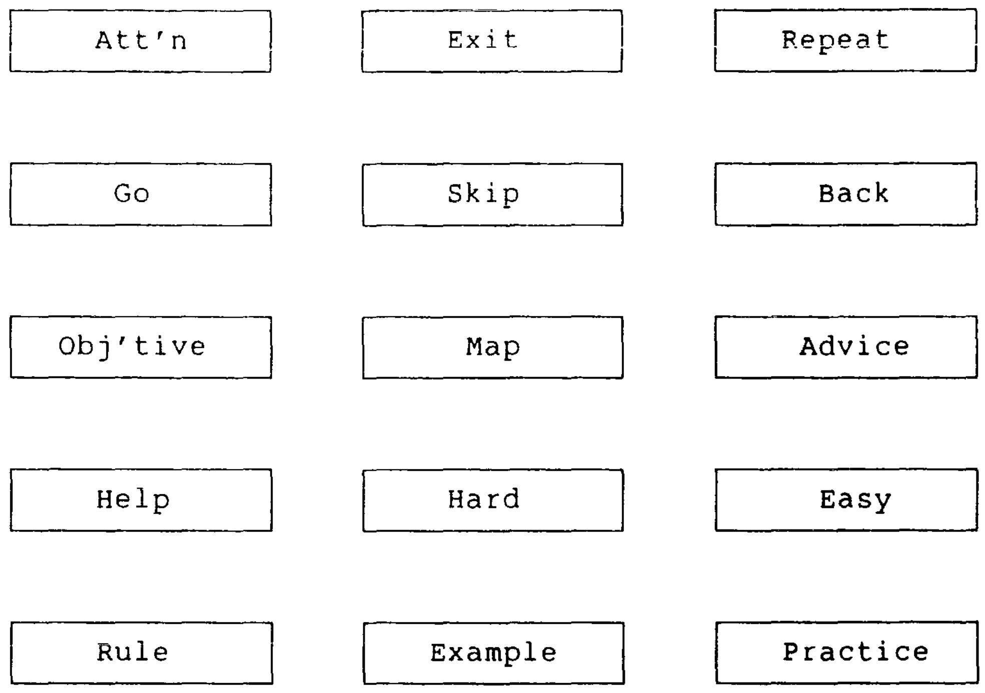


Figure 1.3 TICCIT "learner control" keys

1.2.3 Intelligent Tutoring Systems or Artificial Intelligence in Learning.

Intelligent Tutoring Systems (ITS) are applications in the field of Artificial Intelligence (AI). AI studies intelligence using ideas and methods of computation. However a definition of intelligence seems quite impossible because intelligence appears to be an amalgam of so many information-processing and information representation abilities. AI research started in the mid 1950's with attempts to build intelligent machinery using the human brain as a model. In 1960's AI work focused on the problem solving aspect of intelligence. The General Problem Solver was one of the early systems developed as a result.

Psychologists, philosophers, linguists and others from related disciplines offered various perspectives and methods for studying intelligence. Their contributions in terms of ideas, relationships and constraints gives the some basis and credence that artificial intelligence is in fact possible.

Artificial intelligence offers a new path and methodology to understanding intelligence whose ultimate goal is to make computers 'intelligent'. Using ideas and methods of computation, a new and different basis for theory formation have developed in the artificial intelligence community. Many in this community believe that these theories will apply to any intelligent information processor, be it solid state or biological.[Winsto79]

Most of the theories proposed however are still too incomplete or too vaguely stated (perhaps understood) to be realised in computational terms. In Dreyfus' critique of artificial intelligence he described early efforts on AI which generally followed some pattern. The pattern starts with an early success on simple mechanical forms of information processing, great expectations and then failure when confronted with more complicated forms of behaviour.[Dreyfu72] In fairness research in AI did produce some results in understanding some aspects of the problem solving process. General problem solving strategies were developed and applied. GPS (General Problem Solver) was one such system. GPS was general in that it made no specific reference to the subject matter of the problem. The user has to define a task environment in terms of objects and use operators to apply to those objects. However the generality was restricted to a domain of puzzles like the "Towers of Hanoi". Later AI work recognised that specialised knowledge is required to solve a specific problem. This has led to a more restricted area of domain specific intelligent systems known as Expert systems. Expert systems like MYCIN for diagnosing infectious blood disorders, SOPHIE in debugging electronic devices and DENDRAL for inferencing chemical structure of molecules from mass spectrometry data are the products of decades of research put into AI Intelligent Tutoring systems have benefited from these successes and have either tapped into the existing pool of expert system's knowledge or have used the same ideas in developing its own knowledge.

<u>GENERATIVE CAI</u>

Intelligent tutoring systems or artificial intelligence in computer aided learning has been a recent development. Intelligent tutoring systems aims to apply the idea of heuristics to the field of computer aided learning systems. Generative CAI was mentioned in the previous section on CAL systems, it is the precursor to the Intelligent Tutoring systems.

Generative CAI stemmed both from a practical desire to ease the authors task in preparing teaching material and more importantly, from a different educational philosophy. This philosophy held that students learn better from attempting problems of an appropriate difficulty than from attempting some systematic exposition of material. The Generative CAI method involves writing a computer programme that will generate the material, that is, the problems, solutions and associated diagnostics.

The foreseen advantages of such a system were that:

(a) provision of an unlimited resource of teaching material;

(b) the store taken by teaching material is reduced;

(c) provision of as many problems as the student needs to achieve some level of competence;

(d) ability to control the level of difficulty of problems so that the student is presented with problems relevant to his needs at the time.

SUMMARY

Generative CAL can be summarised as follows:-

$$\begin{tikzcd}
\text{Strategy determines problems} \arrow[d] \\
\text{Problem is generated and presented} \arrow[d] \\
\text{Student solves problem} \arrow[d] \\
\text{Programme solves problem} \arrow[d] \\
\text{Problem solutions are compared} \arrow[uuuu, bend right=90]
\end{tikzcd}$$

Success of the generative model depends on the availability of the task difficulty model, with parameters which can be systematically altered, how could the model then be used to teach say politics or poetry. If the topic in mind can say 'yes' to each of the following questions then it is likely that a generative model can be used:-

(1) Do you have standard format questions?

(2) Is there only one method of solution for each problem?

(3) Can you identify the intermediate steps (assuming you want to comment on these)?

(4) Is it easy to estimate the difficulty of a problem?

(5) Is it easy to find out about what you need to know about the student in order to be able to give him appropriate problems?

(6) Can the different sorts of problems be put in an order of difficulty? (if you have more than one sort.)

The introduction of intelligent tutoring systems raises cautious optimism about boosting the role of the computer aided learning systems. The following section introduces the role of artificial intelligence in computer aided learning.[Sleema82] Intelligent tutoring systems reviewed here are taken from the following subject areas:-

(a) place value arithmetic;

(b) solving simple algebraic equations;

(c) non-deterministic (or backtracking) problem solving;

(d) debugging (of electronic circuits and program/plans);

(e) medical diagnosis.

SCHOLAR

SCHOLAR teaches South American geography and it is regarded as the first CAL programme to be in the category of an expert system. SCHOLAR employed a graph (semantic net) representation for declared facts about geography. The graph contains specific relationships on a domain and do not embody more refined levels of geographic knowledge linked by various changing relationships.

SOPHIE

SOPHIE is a tutor for troubleshooting a piece of malfunctioning electronic equipment, such as a power supply. A simulation package called SPICE which is a non-AI tool and rules which are embedded enabled it to make intelligent use of the simulator. It also provides a natural language type interface to the student, demonstrating the feasibility of good communications between student and the automated tutor.

WEST - How the West was won

"How the West was won" is a computer board game, originally designed at project PLATO. The game gives students drill and practice in arithmetic. The computer board is 70 spaces long and the objective is for the student to be the first player to land exactly on 70. In a turn each player receives 3 numbers, from spinners, which must be used in an arithmetic expression, using the operations addition, subtraction, multiplication and division as well as parenthesis; with the constraint that no operator or number can be used more than once. Special moves are incorporated by adding towns at every ten paces and short-cuts. If a player lands on a town he can advance to the next one. If he lands on a short-cut he can advance to the other end of the short-cut. If he lands on the same space as his opponent the opponent is bumped back two towns unless the opponent piece is on a town.

The coaching environment in WEST is the main element which puts it in the category of an ITS. The coaching system adopts a means of giving appropriate comments based on an idea of "Issues and Examples". A "Focus" or "Breadth" strategy is also used to guide the coach's decision on which issue is to be used on the student. The "Focus" strategy selects the most recent issue discussed between the coach and the student while the "Breadth" strategy selects the issue which has not recently been discussed. There are 3 levels of issue that can be discussed. At the lowest level are the basic mathematical skills that the student is practising. The second level concerns the skills needed to play WEST. Issues at this level are special moves of BUMP, TOWN and SHORTCUT, the direction of a move (eg. Both FORWARD and BACKWARD moves are legal); and the development of a strategy for choosing a move, such as maximising the distance you are ahead of your opponent. The third level deals with the general skills of game playing. One such general skill is the strategy of watching your opponent in order to learn from his moves.

A model of the student's knowledge is kept in relation to an expert player. Giving helpful hints at appropriate moments.

Experiences of WEST from an experiment with 18 student teachers who use the system for at least an hour are noted as follows:-[Burton82] Only 1 did not receive any advice from the coach. 9 teachers commented favourably about the coach's advice. 2 disagreed; one on the basis that the coach's strategy would leave him "vulnerable to attack" which was an element of strategy not known to the expert. 8 out of 10 subjects found the comments helpful in learning a better way to play the game. 9 out of 10 felt that the coach manifested a good understanding of their weaknesses.

In another experiment conducted in elementary school classrooms. Some interesting patterns and results were obtained. The coached students showed a considerably greater variety of patterns, showing that they acquired the skill in using more subtle patterns and not falling into using a set pattern which may prevent them from seeing the relatively rare occasions when some of these moves were important. Surprisingly the coached students enjoyed the game more than the uncoached group. This helps to substantiate that the coaching principles were well developed and did not destroy the enjoyment of the game.

GUIDON

GUIDON grew out of one of the best known expert system, MYCIN. MYCIN is a rule based expert system on the treatment of infectious blood diseases. GUIDON has a teaching expert and a model of the student's performance.[Clance82,Clance84]
GUIDON utilises MYCIN's set of knowledge rules called 'production rules' which constitute the MYCIN knowledge base. The MYCIN knowledge base described by Clancey contains about 450 such rules as well as several hundred facts and relations stored in tables, which are referenced by the rules. Each of the rules consist of a premise which, if true, justifies the conclusion made in the "action" part of the rule. An example of a MYCIN rule is shown below:-

If (1) the gram stain of the organism is gram negative, and (2) the morphology of the organism is rod, and (3) the aerobicity of the organism is anaerobic, THEN there is suggestive evidence (0.6) that the genus of the organism is Bacteroides.

Figure 1.4 Sample MYCIN rule.

GUIDON added two other levels to strengthen the performance of the MYCIN's 'production rules'. One level is used to justify individual rules, and another to organize the rules into patterns. The teaching expertise of GUIDON is independent of the domain knowledge base and is used to carry on a tutorial type dialogue to present the domain knowledge to a student in an organised way.

The separate interacting components of GUIDON provides the basis of an intelligent tutoring system.

A Self-improving Quadratic Tutor.

The "Self Improving Quadratic Tutor" teaches quadratics based on the discovery or problem solving method of teaching style. The system starts by carefully choosing a set of equations for presentation to the student who will then try to determine the solution. The set is chosen specifically to enable the student to discover the general rule for solving this class of problem. Once the student appears to have found a particular rule, the "Tutor" will present more examples, some and others to challenge that rule so that it is refined and generalised.

O'Shea has expressed the decisions about the teaching strategies in the form of production rules that guide the generation of examples.[O'Shea82]

SPIRIT

SPIRIT is an intelligent tutoring system for teaching probability theory. It is a later development and tries to provide a complete intelligent tutoring environment. The system manages a unique flexible tutoring style. On one hand the system may behave as a tutor who mostly observes the student without interference, intervening only when things are really going wrong, and on the other hand it may behave as a tutor who manages a "questioning and answering" type of dialogue. Based on a belief constructed about the student's aptitude, the system frequently changes its tutoring style. SPIRIT integrates several artificial intelligence methods that include a theorem prover, a production system, an object oriented system and procedural knowledge embedded in LISP code.[Pople84]

1.3 Conclusion

This chapter aims only to cover some aspects of the emergence of CAL systems. There are still numerous approaches in CAL that are not mentioned but are more comprehensively covered in the following books:-

"An Introduction to Computer Assisted Learning" by
P. G. Barker and H. Yeates, Prentice Hall, 1985.

"Learning and Teaching with Computers" by
J. Self and Tim O'Shea, Harvester Press, 1983.

"Intelligent Tutoring Systems" by
D. Sleeman and J. S. Brown, Academic Press, 1982.

As a result of the early attempts at providing the educational assumptions and CAL systems many useful lessons have been learnt. Well known systems have been adapted for the commercial market, for example, PLATO concepts have been adopted and released on microcomputers like the ATARIs' and IBM PCs'.

References

Atkins69.
Atkinson, R.C. and Wilson, N.A., Computer Aided Instruction, pp. 5-6, Academic Press, 1969.

Bitzer76.
Bitzer, Donald, "The Wide World of Computer-Based Education," in Advances in Computers Vol 15, ed. Morris Rubinoff & Marshall C. Yovits, pp. 239-283, Academic Press, 1976.

Bitzer70.
Bitzer, Donald and Skaperdas, D., "The Economics of a Large Scale Computer Based Education System : Plato IV," in Computer Assisted Instruction, Testing and Guidance,, ed. Wayne Holtzman, pp. 17-29, Harper and Row,New York 1970, 1970.

Burton82.
Burton, R. R. and Brown, J. S., "An investigation of computer coaching for informal learning activities," Press, 1982.

Clance82.
Clancey, W. J., "Tutoring rules for guiding a case method dialogue," in Intelligent Tutoring Systems, pp. 201-222, Academic Press, 1982.

Clance84.
Clancey, W. J., "GUIDON," Proceeding of the Joint Services., pp. 181-187, (U) Denver Research Inst. Co., June 1984.

Dreyfu72.
Dreyfus, Hubert L., What Computers Can't Do : A critique of Artificial Reason, Harper & Row, 1972.

Nilsso80.
Nilsson, Nils J., Principles of Artificial Intelligence, Springer Verlag, 1980.

O'Shea82
O'Shea, T., "A self improving quadratic tutor," in Intelligent Tutoring Systems, pp. 309-334, Academic Press, 1982.

Pople84.
Pople, Harry E. Jr and Barzilay, Amos, "SPIRIT: an evolutionally design intelligent tutoring system," University of Pittsburgh, University of Pittsburgh & XEROX Palto Alto Research Center, July 1984.

Self83.
Self, John and O'Shea, T., "Learning and Teaching with Computers," The Harvester Press, 1983.

Sleema82.
Sleeman, D. and Brown, J.S., "Introduction: Intelligent Tutoring Systems," in Intelligent Tutoring Systems, pp. 1-8, Academic Press, 1982.

Winsto79.
Winston, Patrick H., Brown, Richard H., and Mike Brady, Artificial Intelligence: An MIT Perspective, M.I.T., 1979.

Yeates81.
Yeates, H., Some Experiments in Computer Aided Learning, M.Sc Thesis, University of Durham, 1981.

CHAPTER 2

TOOLS FOR A CAL SYSTEM

2.1 Introduction

The first thing a carpenter would do when he is building a new house or furniture is to collect his tools. The tools that he selects would depend on the type of task that needs to be carried out. The same should be true if a designer/programmer is developing a new computer application. Though the tools used by a designer/programmer may not be as tangible as a hammer or a screwdriver they nevertheless satisfy the functional principle of tools. Tools not only make the task easier but also increase the productivity of the user.

Each type or class of application has its own particular tool which suits it best. This chapter takes a look at the computer and its tools used in the development of computer aided learning systems.

2.2 The Computer as a tool

Computers have come a long way since ENIAC (Electronic Numerical Integrator and Computer), the first electronic computer in 1946. The extent of use has spread from scientific, military, commerce to education and recreation. The range and variety of computers are wide and bewildering. Two main types of computers are in use today, the analogue type and the digital computer. The digital computer operates with numbers and letters of the alphabet represented as numbers whereas the analogue computer calculates on a different basis. Analogue computers are far less generally used, much less versatile than the digital computer. In this thesis, the computers talked about will mean to describe the digital type.

All present day computers operate on stored programme of instructions and in general computers have four basic features

1) MEMORY - or store which holds the information whether data or programme instructions.
2) INPUT - is the channel for receiving information.
3) OUTPUT - is the channel to release information.
4) PROCESSOR - for manipulating the information and controlling the overall operation.

Basic components of the computer are shown in the following diagram:

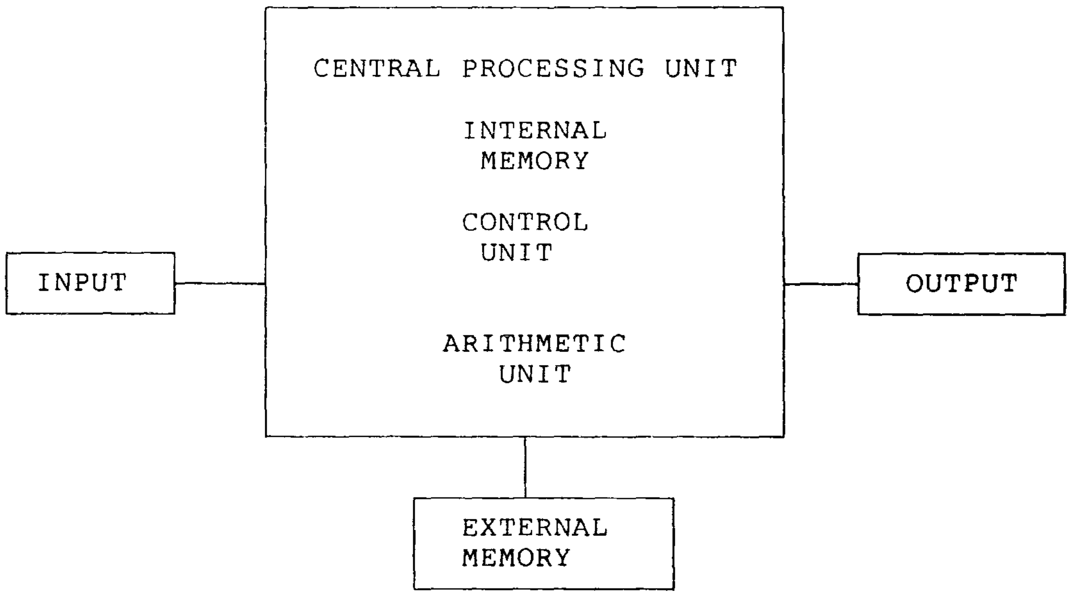


Figure 2.0 Basic components of a computer.

2.2.1 Memory

Memory on the computer can be placed in two distinct categories - internal and external.

Internal memory or core memory can be directly accessible by the processor. The external memory on the other hand can only be accessed indirectly via some auxiliary control unit.

Internal memory usually consists of either

1) RANDOM ACCESS MEMORY (RAM for short), or
2) READ ONLY MEMORY (ROM).

RAM is different from ROM in two main areas, RAM requires power to maintain the information contents and its contents can be erased and changed by the processor. The ROM on the other hand does not need power to return its information and its contents are normally static or permanent. There are variations of ROM like EPROM which though does not need power to maintain its memory contents, allows its contents to be changed. The processor will require the relevant data and/or programme to be loaded in the internal memory before it can perform any operations.

External memory is used as a backing store and can hold vast quantities of information. The principal types of device used for external storage are:

1) magnetic tape units and
2) magnetic disc unit.

Magnetic disc units are much faster than the tape units. It also has the advantage of direct access to a particular piece of information. In contrast tape units must sequentially skip through preliminary information until it has reached the location where the required information is residing.

2.2.2 Control and Arithmetic

The control unit must co-ordinate the actions of all the various parts of the computer. For example instructions transferred from the internal memory to the control unit are decoded and control signals are sent to other units.

The arithmetic unit not only performs calculations but also executes logical functions such as testing the signs (+ or -) of numbers, shifting parts of numbers and moving the numbers between storage locations. It can also be employed in modifying the programme itself by altering the instructions held in main store. The ability to modify programmes is the key to the computer's flexibility and hence its wide applicability.

2.2.3 Input and Output

The input and output of information to and from the computer is achieved by I/O (Input/Output) devices or peripherals. The keyboard is one of the most common means of input. Other popular devices are light pens, 'mouse', touch screen, track ball etc.

Output devices like CRT (Cathode Ray Tube), LCD (Liquid

Quartz Display), Gas plasma are some of those grouped under visual display units. Such devices display information only. Printers are another category of output devices used if a permanent record of information is required.

The great variety of equipment for putting data into, and taking data out of, computers allow users to do things in many different ways. This wealth of choice creates more opportunities for new applications. The computer has become thus a most versatile and productive tool.

2.3 Software Tools

Software is used to describe the programmes a computer needs to do its job. The dictionary describes a tool as anything employed in performing an operation. Software tools can be very generally put as - programmes used to perform the operation of solving a data-processing problem. There are two categories of programmes - systems and user programmes. A user programme is aimed at making the computer do a specific job, such as keeping an inventory list or controlling a machine tool. Every user programme is written in a language which could eventually be understood by the computer. The process of translating the user programmes into computer readable form is done by compilers. Compilers are themselves programmes which form part of a suite of programmes categorised under system programmes. System programmes function to provide a better environment for writing, testing, running and storing programmes. They form thus the basic set of tools for the computer user. In practice, operating systems, editors programming languages, programme libraries are considered as system programmes.

2.3.1 Operating Systems

An Operating System is a 'layer' of software designed to 'insulate' the user programmes from the hardware of the machine. The functions performed by the operating system are often different but interlocking, which often makes it large and complex. The UNIX operating system, for example, originally consisted of about 100,000 lines of code mostly written in a language called C and some in a lower level assembly language. [Thompso75] Some of the important functions performed by the operating system are:

1) Control of peripherals: The operating systems takes care of the peripherals on behalf of the user programmes. This eases the programmer's work and worries, for example the spooling function which spools the output to, say a magnetic disc for subsequent printing.

2) Job Control: All user work on the computer is carried out by jobs or processes. A job or process is a single

sequence of events and consists of some computer memory and files (internal and external) being accessed. The single sequence of events e.g. might be a high-level language programme written, must first be translated into machine code by a compiler then followed by the execution of the machine code programme.

The job control portion of the operating system will guide and schedule jobs through the right sequence of steps according to the user's job description submitted.

3) File control: The operating system controls the external store, to ensure that every user can store his own information and retrieve it, and that different users cannot interfere with one another's data.

4) Software control: System software like assemblers, compilers, editors, utility software libraries are managed by the operating system to allow immediate availability when required. The editor e.g. might be used in conjunction with the file control to change the contents of the filed information.

5) Provision of multi-access: A multi-access computer is one which serves many people at the same time, each one using a console with a keyboard and visual display unit. The operating system is responsible then for sharing the system

resources like CPU and store among the users and ensures that users' processors do not clash.

6) Accounting: The system keeps track of the resources used by the users. This information could be used to issue bills for the resources used or might be used to study the system usage trends for tuning the system performance.

The Operating System is the computer's most basic software tool that other higher level software depends upon. The performance of the operating system will be crucial for the successful implementation of other tools.

2.3.2 Editors

Editors are programmes which allow computer files such as programmes or text to be modified. Two main classes of editors have merged, one is the line or context editor and the other is the screen editor.

Context editors emerged during the days of slow mechanical terminals such as the Teletype model 33 and 37 teleprinters. Generally context editors are different in the text manipulation facilities they provide and their command syntax. The type of text facilities normally required by the user will more often than not be present in most context editors implemented on different machines. These basic facilities would include:

SEARCH	-	to find an occurrence of a string;
INSERT	-	to insert a string at a given point;
DELETE	-	to delete a string or line at a given point;
MODIFY	-	to search for a string and modify it to a prescribed new form;
END	-	to end the editing process and restore the newly corrected document in the file store.

Screen editors become more readily available when video terminals are used as the display units. Video terminals could display more than a line at a time, a common feature is a 24 line by 80 character per line screen. A full-screen editor allows most or all of the display screen to show a continuous group of lines from the file being edited. Unlike the command line entry required to enter and specify the editing requirement, most of the entry will usually be in the form of a single or a combination of key presses.

A screen editor has the advantage of the user being able to see actually what is being done to the text at almost the same instant. However screen editors might not replace context editors in situations whereby large systematic changes are required.

2.3.3 Files, Databases and DBMS

Here we will discuss on the concepts and tools that are used to manage the information reposited in a computer system.

How is information stored in a computer?

Storage at the basic level represents binary digits, or bits. Physically the storage location containing a bit is capable of only two states: "on" and "off". The "on" state represents the binary digit 1 and "off" the binary digit 0. All information in the main storage as well as external storage mechanisms, like discs and tapes, is composed of 1's and 0's. A grouping of bits that form a location capable of holding some information e.g. a character 'A' is known as a byte. The number of 'bits' that grouped into a byte varies from computer to computer. In most systems a byte consists of eight bits, some machines like ICL 2930 and ME29 have a 6 bit byte. On some computers the byte is the fundamental unit of the computer's main storage, most computers also use a larger storage unit consisting of a fixed number of bytes called words. Just as the size of a byte can vary from computer to computer so can a word.

Table 2.4 below shows the different byte, word sizes of some machines.

Machine	byte size (bits)	word size (bits/bytes)
INTEL	8	8
INTEL 80286	8	16/2
ICL ME29	6	24/4
VAX 11	8	32/4
IBM 370	8	32/4
ICL 2980	8	64/8

Table 2.4 Machine Byte/word sizes

Data held in many storages usually consists of characters or numbers. A character generally requires one byte of storage. To represent a character in a byte, such as the letter A, a pattern of 1's and 0's is used.

Most computers receive information in the form of a stream of characters flowing from a keyboard for example. A common keyboard can normally produce about 90 different characters, including some special ones like end-of-line character. A fixed-length binary code e.g. a byte is sufficient to represent a combination of characters since $2^6 = 64$ and $2^7 = 128$.

Two well-known character codes types are the ASCII (American Standard Code for Information Interchange) and

EBCDIC (Extended Binary Coded Decimal Interchange Code). In ASCII code, an A is represented by a pattern of 7 bits, as follows,

1000001

In EBCDIC the A is encoded as

11000001

Numbers usually need a word or more of storage. Numbers are represented in binary in the computer. A 16 bit word computer would have a limit on its largest whole integer. A longer word size will result in a higher limit on the integers it can handle.

DATABASES

Managing data as a resource has lead to the emergence of database technology. The main arguments against the use of conventional data files approach are:

1) Data Redundancy. This happens when the same data item is recorded in multiple files. The effects are that storage space is wasted, more updates are required to input or change the same data item in different files and as a result inconsistencies usually result.

2) Inconsistent Data. With the same data item to be kept on multiple files it is difficult to maintain the data

consistently. The result might be errors in the application programme e.g. when an address of a person is updated in one file but not on another file might cause problems if some mail is directed to the person from the processing of the two files.

3) Limited sharing of data. Most files are grouped by their applications and used within the application. As an example, a payroll file containing employee particulars may be used by a Finance department and the data may be duplicated in another file used by the Personnel department.

4) Lack of Controls. It is easy to imagine how duplicated files and data items could get out of control with inconsistency and redundancy not checked and no inherent checks on the data.

5) Low Productivity. In conventional file processing systems, programmers must design each record and file used by a new application programme. This hindering process not only plagues the programmer during the development phase but also during the maintenance phase. When modification to a file is performed it often requires that the programme or programmes affected be modified.

The disadvantages have been offset slightly by the emergence of a number of support packages e.g. packages which support data dictionaries. The trend has been from a data file storage approach that is dependent on physical data structures, to data base management systems with physical and logical data independence.

The Data Base approach aims to minimise the shortcomings of the traditional file processing approach. A database can be defined [MARTIN81] as "A shared collection of interrelated data, designed to meet the needs of multiple types of end users. The data are stored so that they are independent of the programmes that use them. A common and controlled approach is used in adding new data and modifying and retrieving existing data. A database is not only shared by multiple users, but it is perceived differently by different users".

Three types of approach to database structures have emerged that reflect on how computer information is managed in relation to the real world information.

They are

1) Hierarchical
2) Network
3) Relational

Hierarchy or trees are familiar structures for example in representing the positions of authority of an organisation. A hierarchy consists of elements, called nodes, the uppermost level of the hierarchy has only one node called the root. Except for the root every other node has one node related to it at a higher level, and this is called its parent. Each element can have only one parent but many lower level elements known as children:

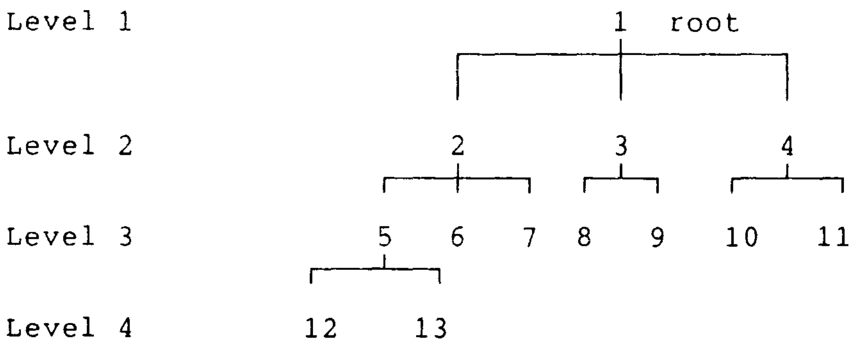


Figure 2.1 A Hierarchy

Network or plex approach is different from the hierarchical one on the basis that one child can have more than one parent, e.g.

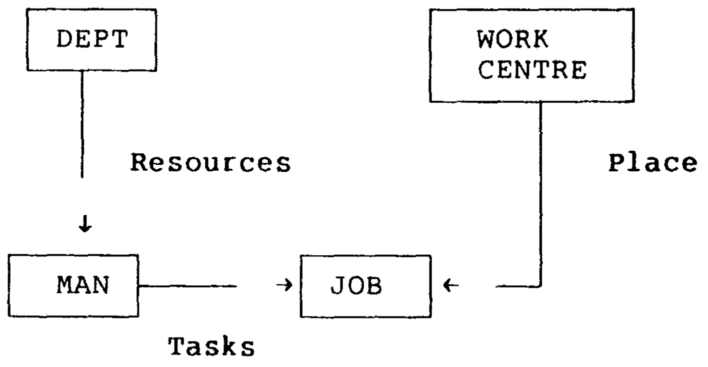
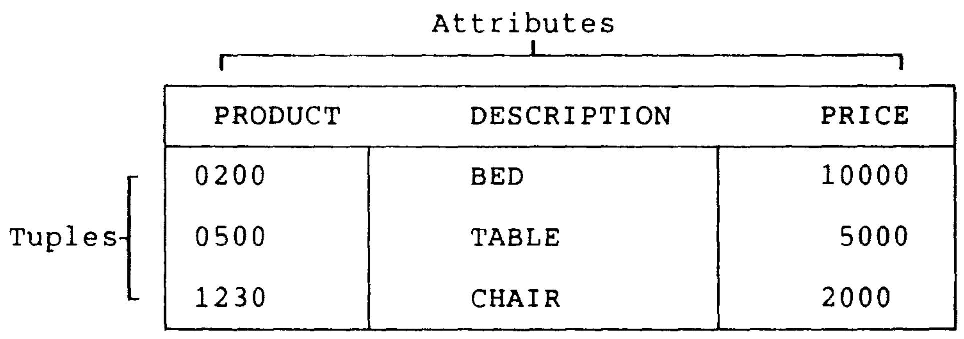


Figure 2.2 A Network

The relational approach to databases is based on the mathematical theory of relations. The approach thus uses relational calculus terms to describe the databases and operations on the data.

Data in a relational database is stored in a tabular form known as a relation. Each distinct data item is known as an attribute value. A tuple is a collection of values that compose one row of a relation. Figure 2.3 shows a relation.

A domain is a set of possible values for an attribute.


Attributes
Tuples

PRODUCT	DESCRIPTION	PRICE
0200	BED	10000
0500	TABLE	5000
1230	CHAIR	2000

Figure 2.3 PRODUCT RELATION

E.F. Codd contributed the significant papers on the relational model and also proposed the concept of normalisation. Normalisation is defined by Codd as a step by step reversible process of replacing a given collection of relations by successive collections in which the relations have a progressively simpler and more regular structure. The aim is to find relationships between data that are free from undesirable interactions, and so simplify the process of maintenance and data retrieval.

The prime objective of normalisation is the production of an ideal data format known as Third Normal Form or 3 NR for short. The stages to 3 NR are:

```
UN-NORMALISED        =    CHAOTIC
                     |
                     |     Remove repeating groups

                                      ↓
1st Normal Form
                     |    Ensure all data fields are
                     |    dependent on all key fields

                                      ↓
2nd Normal Form
                     |    Remove transitive dependence
                     |

                                      ↓
3rd Normal Form      .
                     .
Boyce Codd Normal    .    All field depend on Prime Key
                     .    Remove binary - join dependency
                     .    that
4th Normal Form      .    does not imply functional
                          dependency.
```

Boyce-Codd Normal forms and the 4th Normal forms are later additions to the third Normal forms.

DBMS

After the normalisation process is completed the data can then be organised for loading into a database. The tools that are used in:-

1) creating and updating the database;
2) querying the database;
3) generating reports from the database;
4) maintenance of the database;
5) administrating the database;

and the database itself, are collectively known as a database management system or DBMS for short. DBMS adopts one of the approaches to the management of database system, that is, hierarchical, network or relational.

IMS is a DBMS running on IBM main-frames that supports the hierarchical view of data. IMS has an INSERT (INSRT) command for inserting a record into its database. DL/1 retrieval sublanguage is a navigational method of extracting data in an IMS database [Bradle82].

The Network or CODASYL (Conference on Data System Languages) DBMS were based on the reports in 1969 and 1971 by the Data Base Task Group (DBTG). The Programming Language

Committee under the CODASYL executive committee went on to incorporate a DBTG Data manipulation language (DML) and a subschema into the COBOL's data division. The DML would become part of COBOL's procedural division language. Commercial implementations of the Network DBMS are TOTAL and IDMS.

The relational approach developed largely in the IBM research laboratories at San Jose, California. E.F. Codd contributed the early significant papers on the subject.

All files in the relational database are viewed logically as simple tables(like the table shown on Figure 2.3). To manipulate the data in a relational database, E.F. Codd developed a non-procedural language, consisting of mathematical expressions, called DSL(Data Sub-Language) APLHA. A more English like alternative Data Sublanguage to DSL ALPHA is SQL(Structured Query Language) which was developed by IBM for their relational database system known as System R [Codd70].

INGRES is a relational DBMS developed at the University of California at Berkeley running on UNIX operating system. INGRES has a relational query language modelled after the ALPHA data-sublanguage called QUEL. Another tool for information retrieval in INGRES is EQUEL which allows QUEL statements to be embedded in a programme (BASIC, C, COBOL, FORTRAN and PASCAL).[Relati54] [Held75]

Another query language called CUPID (Casual User Pictorial Interface Design) has been implemented to access information in an INGRES database. CUPID is different from other query languages in that queries are written by aid of pictorial symbols. Once the query picture has been drawn using a graphics modelling system (PICASSO), an output string is produced by PICASSO which is compiled into QUEL using UNIX's YACC [McDona74].

Another form of query language that accesses a relational database is one called Query by Example. Query by Example allows queries by entering an example of a possible answer in the appropriate place in an empty table. Each operation is specified by using one or more tables built up on the screen, with column names being supplied by the system and other parts by the user.

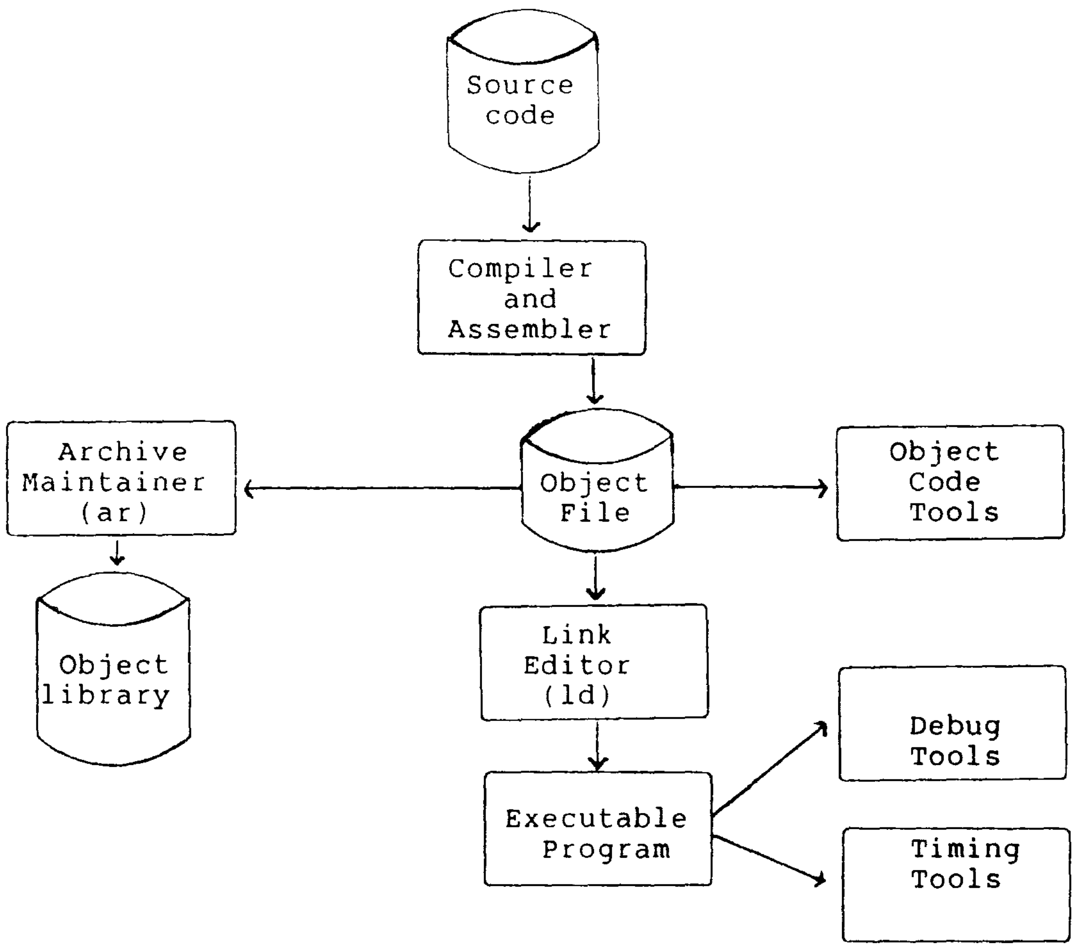
This form of query is novel in that the user need not resort to extensive use of data dictionaries, because it is the system who supplies some information to the user. [Zloof75].

The numerous facilities provided by the DBMS, and the data administration needed, mean that initial overheads like storage space (programmes), training and data administration have to be taken care of before the longer term benefits of quicker turnaround time in report generation, better controls and security can be realised.

2.3.4 Programming Tools

Programming languages, libraries, debuggers, programme verifiers, editors, etc., are some of the software tools used by programmers to aid the development of software. Some of these tools are essential (programming language) while others are helpful (programme verifiers). During the batch processing days, a programmer would normally code his programme onto a coding sheet, send it to be key punched on cards, send a job sheet for the operators to load/compile the programme (on cards), quite often examine the compilation error listings, correct the errors on the coding sheets and the process recycles. (Testing and debugging the actual runs still had to follow). Currently editors allow the programmer to key the programme straight into a computer file, perhaps using a programme verifier to perform a preliminary check on the programme before compilation. The initial set of tools have cut down the time consuming and tedious process required in the batch processing days. The UNIX system has been singled out as one example system that provides a rich set of programming tools. The UNIX tools have been emulated in non UNIX environments, an indication perhaps of the success of the UNIX environment.[Kernig78] [Kernig76] [Snow78]

Major UNIX object Code Programming Tools



```
Commonly used tools to works with object code on UNIX

adb  - general debugger;       od - octal dump;
dbx  - source level debugger;  nm - print name list;
prof - display profile data;   strip - strip relocation data;
size - display size of object file;
ar   - build and maintain libraries;
time - time a command;
```

Figure 2.4 UNIX Programming Tools

2.4 CAL Tools

This section describes the authoring languages and the user interface tools that affect the preparation and presentation of CAL material.

2.4.1 Author Languages and Authoring Systems

Writers of CAL lessons have to use either a programming language or an author language to write the CAL lessons. CAL writers or courseware authors have the following major tasks to perform:

1) deciding upon the subject matter to be presented;
2) structure the lessons to be presented by the computer;
3) decide on the instructional strategy to be used e.g. drill practice, tutorial;
4) select the media on which the lessons are to be based e.g. graphics, text;
5) if necessary or possible, specify the evaluation criteria for the performance of the lessons and the students.

[Barker85]

General purpose programming languages can be used to implement a wide range of computer applications and as an added attraction give relative portability. Author languages in a sense are high level but more specifically aimed at those preparing computer based learning material. The necessity of learning how to programme in order to develop courseware discourages to some extent teachers who will attempt to use a computer as part of their teaching activities. Development time and resources can be reduced when author languages or authoring systems are used. A commonly accepted rule of thumb is that between 50-200 hours of development time are required for each hour of instruction developed. This time includes analysis, design as well as the programming, debugging activities. Thus it can be argued that if creating a suitable tool or environment which eliminate or sharply reduce the time taken for the activities will significantly decrease the development time and hence the cost of the courseware.[Kearle82]

Author languages generally provide facilities to:-

1) present the material via frames;
2) test student's responses;
3) provide branching strategies for remedial/reinforcement material.

Author Languages are an alternative way of preparing CAL material to the general purpose programming languages. Their function is to ease the work of the author by providing specialised functions and facilities for the authors to cater for operation required in the CAL situation. Thus author languages are a class of special purpose, higher order application languages which facilitates the writing of CAL material.

Another aspect to be discussed here will be authoring systems which, though historically derived from author languages, represent a high level interface intended to allow authors to create CAL material without having to learn or use a programming language. In short, author languages make it easier for programmers to develop the CAL material, and authoring systems makes it easier for non-programmers. Productivity has been measured to increase by three times when using an author language and about fifteen times when using an authoring system with on-line editing facilities.

2.4.2 Types of Author Languages

The range and variety of languages used in CAL is very wide, Hoye and Wang (1973) list 62 different languages used, 16 were assembly level, 46 general purpose. Here we will look at some of these languages.[Hoye73]

1. TUTOR

TUTOR is the language used to construct PLATO lessons. The initiator of the TUTOR language is Paul Tenczar. The language evolved as it was developed in a user environment adapting to the user's needs.

The TUTOR language statement appears on a command plus a tag field, over 160 commands are available in the language. The commands fall into four categories:-

1) Display;
2) Control;
3) Calculation;
4) Judging.

Some examples of the commands are

Display Command	Description
WRITE	to write on display
AT	where to write
DRAW	to draw line figures
CIRCLE	draw a circle

Control Command	
JOIN	to join another part of lesson
JUMP	to go to another part of lesson
GOTO	loop

Calculation Command

CALC	allow author to write almost any computation statement in the tag field

Judging Command

ANSWER	permit student to answer
STORE	student answer to be stored and processed.

The attraction of TUTOR language is its ability to handle interactive graphical images. These images are produced on high resolution touch-sensitive screens attached to special terminal devices.

2. COURSEWRITER

COURSEWRITER is an author language initially developed by IBM (International Business Machines) for their main-frame IBM 360. The language is able to support multi-media devices like slides, audio equipment and text images.

The language uses mnemonics and is claimed by the designers to be easy to learn both by novices and experts. An illustration of a COURSEWRITER lesson is shown in figure 2.5. Contrary to the designer's claims, some teachers have found it both complex and inflexible.

```
Question 1

qu    What is the sum of 10 and 5?

ca    15

br    question 4

wa    50

ty    No, the answer you gave is the product
      of 10 and 5.  Try again.

      Make sure that you have typed a numeric
      answer.
      Try again.
```

Command	Reaction
qu	question to student - await a reply
ca	anticipated correct answer
ty	type out
br	branch
wa	anticipated wrong answer
un	unanticipated answer

Figure 2.5 Example of COURSEWRITER lesson

3. TICCIT

About the same time PLATO was developed, TICCIT - Time-shared Interactive Computer Controlled Information Television was also being developed.

The system combines minicomputer and television technology, and produces a highly individualised learner controlled instruction for as many as 128 students. This is achieved through the use of computer generated colour displays, videotape, audio facilities and a specially designed interactive keyboard.

Authoring on TICCIT is via a series of special authoring forms. The authoring forms may be computer generated using on-line courseware production at a VDU terminal, or they may be of the conventional paper variety for off-line use. The various forms act as templates that permit the author to enter the learning rules, instances, examples, colour of the learning frame, as well as the display characteristics. These features allow the author to produce courseware without the need for computer programming.

4. UNIX LEARN

In some respects UNIX LEARN's author language is similar to TUTOR in that it uses English like commands (See fig 2.6). The range of commands is limited to about 17 in the original system [Kernig79]. The commands can be categorised into these areas:

Display	- #print
User response	- #user #copyin #uncopyin
Testing user response	- #cmp #match #bad #succeed #fail
Logging	- #log
Branching	- #next

UNIX commands and programmes can also be called from the author language.

These features could allow the author to use only existing tools available on the system to be incorporated into the text. Though the author language is sufficient to allow easy composition of lesson it assumes that the author is familiar with some basic UNIX concepts and use of UNIX editors to compare and edit the scripts.

Familiarity with the UNIX system and the local programmes available for some specialised use, is vital if the author wishes to write courseware for difficult subjects like graphics or a programming language. The on-line documentation ('man' facility) for LEARN did not really elaborate on the uses of the LEARN commands which is another setback for those wishing to write LEARN lessons for the first time. There is scope for improving the user's ability to writing LEARN lessons.

```
#print
The "ls" command will list the names of the files
in your directory.  Is there a file named "junk"
present?  Find out and then type "yes" or "no".
#copyin
#user
#uncopyin
#match no
#log
#next
2.1a 10
2.2a 5
```

Figure 2.6 Example of Learn author language

5. PILOT

PILOT was developed in the early 1970s by John Starkweather in the USA. It was based on an Intel 8080 system hence its many new implementations have become available on a variety of micro computer systems[Barker85][Starkw69].

PILOT has the potential of being used more widely since it has already a large base of CAL materials on a large number of microcomputers. Though several ad-hoc standards exist they could form the basis of a standard. The number of commands to be learnt is small, about ten or so, but extensions have been added to this basic set of commands to allow for sound and graphics.

2.4.3 User Interfaces

E.B. James proposed a dialogue type interface which guides the user along a path of questions and answers given an initial response by the user. The dialogue interface mentioned in E.B. James' Science Museum terminal project aims primarily at novice users. Interesting observations from the Science museum terminal project were:

1) users learn to use the system often through seeing how others use it;
2) users tend not to use written material until difficulties are encountered;
3) restrictions imposed on the users by the system in that users had to adapt and answer the systems' queries [James81].

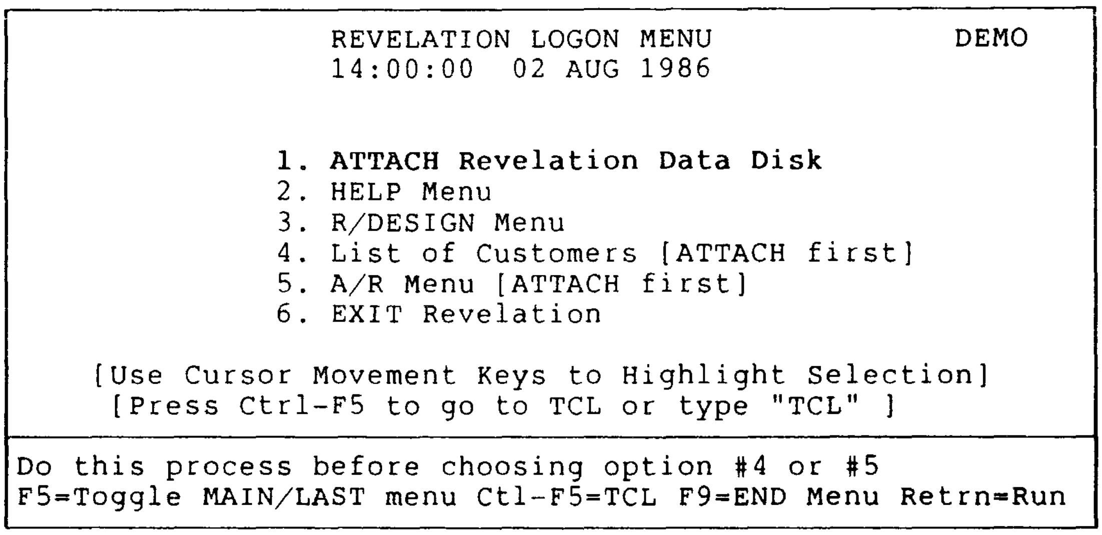
Command line Interface

Using the video alphanumeric keyboard terminal as the physical interface, the most common form of interaction between a user and the computer is usually facilitated via programmes which allow tasks to be sent to the computer by typing and sending a line of commands. Command-line type interfaces have commands in a word or mnemonic form that describe their function. For a novice user, adapting to this form of interface can be difficult, since there is both a need to be able to recall the right command needed, and the accompanying syntax of the command.

One proponent (E.B. James) of a more friendly interface has used the phrase 'protective ware' to signify software that shield the user from the harsher aspects of the operating system.

MENU Interface

Another common user interface that is used is the menu type interface. The user will be presented with a series of options that are numbered like a menu. Selecting the options is either done by typing in the number or name of the menu option corresponding to the option given in the menu. This form of interface guides the user by limiting the choice and giving a more detailed explanation of what the options do. Figure 2.7 presents an example of a menu type interface taken from the logging on menu of a database package called Revelation.



```
                  REVELATION LOGON MENU             DEMO
                  14:00:00  02 AUG 1986


               1. ATTACH Revelation Data Disk
               2. HELP Menu
               3. R/DESIGN Menu
               4. List of Customers [ATTACH first]
               5. A/R Menu [ATTACH first]
               6. EXIT Revelation

     [Use Cursor Movement Keys to Highlight Selection]
      [Press Ctrl-F5 to go to TCL or type "TCL" ]

Do this process before choosing option #4 or #5
F5=Toggle MAIN/LAST menu Ctl-F5=TCL F9=END Menu Retrn=Run
```

Figure 2.7 Example of Menu Interface from Revelation System

The criticism of being unduly restrictive is levied at menu-type interfaces, a user may need to select options from two or more menus before getting to what is required. Most menu-driven systems thus have incorporated the option of command line entry to give users more flexibility. [COSMOS85]

Dialogue (Natural Language) Interface

Dialogue or Natural Language Interface is another form of communicating with the computer system. It aims to be more flexible and user friendly by:-

(1) allowing the use of English like words and sentences to be used in phrasing of a request, query or answer;

(2) initiating a dialogue either in the form of a question or by providing a meaningful answer if the dialogue programme could not cope with the situation[NCC80].

USERTAB is an example of a natural language based user interface. It is used mainly for report generation information retrieval purposes, and run on a variety of minis and main-frames:- IBM 360, ICL 2900's, SPERRY UNIVAC 1100 and ICL ME29. Retrieving information, or generating reports from existing data, traditionally requires a programme to be written, for example in COBOL. This requires resources of time and effort by the user and programmer, as well as computer time for editing, testing, debugging and running the programme. Using a natural language type interface like USERTAB can decrease programming time by training users to phrase their report or query requirements directly to the computer via the use of an English like language. The following example shows how USERTAB language is used to generate a report from a conventional file named "FILEA". USERTAB also allows access to IDMS database in addition to conventional files.

```
READ FILEA

REPORT 'PARTS STORED IN BIN AREAS 5, 6 AND 7'

REJECT UNLESS BINAREA IS IN THE RANGE '5-7' AND SORT

BY PARTNO WITHIN BINAREA.

PRINT BINAREA, PARTNO AND QUANTITY.

FOR EACH BINAREA AND FINALLY, TOTAL THE QUANTITY AND

CALCULATE AVERAGE = QUANTITY DIVIDED BY COUNT.

PRINT QUANTITY, AVERAGE AND COUNT.
```

Figure 2.8 USERTAB Example

Command-line, dialogue and menu-type interfaces can be grouped under textual or keyboard interfaces. A different approach to user interfacing is known as icon or window-based, which depends heavily on graphical displays, windows and usually joysticks or mouse-type input devices .

Visual or Graphical based systems.

Hardware and software are integral in providing the user-machine interface for a CAL environment. Certain known structures in CAL need some form of specialised presentation which need certain equipment not normally catered for. For example graphical displays need visual display units or graphics capability printers.

Use of graphical displays for CAL has been and will continue to be an area of interest. Graphical requirements and standards have grown over the years. Around 1977, typical requirements of graphical display terminals were

1) terminal line speed should be at least 1200 baud,
2) resolution around 320 x 256;
3) Input and output facilities, like light pens, tracker ball; software to transmit screen position back to computer[Shirle78].

Currently,

1) terminal line speed can be at 1200 (over telephone) but higher speeds of 9600 and even to 19,200 (Blit) are not uncommon.;
2) resolution could range from 720 by 348 (Hercules) to 800 by 1024 (Blit terminal). [Pike84][Hercul85]

"A picture is worth a thousand words" is very often quoted to emphasise the advantage of pictorial rather than textual displays to present some piece of information. In some situations, like showing the locomotion of an object or graphs, no words can adequately describe the contents. The ability to present pictures can be thus be a powerful tool for CAL.

The earliest work on this form of user interface can be traced back to Douglas Englebert's work on using computers to augment human intelligence. The use of a mouse-device for input, and the incorporation of multiple windows into the design of text editors were introduced by Douglas Englebert. Early significant contributions also came from Xerox PARC (Palo Alto Research Centre) and its Learning Research Group. [Ingall81] [Warfie83]

Steve Jobs later implemented the research done at Xerox PARC's SMALLTALK project on a commercial basis, with the introduction of the Apple Lisa microcomputer, which supports a SMALLTALK-like environment. The features of this environment are a heavy dependence on graphics to manage the windows, objects (icons), and a complementary input device, usually a mouse, to select the objects (icons) for further processing. Though the Apple Lisa was not as successful commercially the ideas caught on. Chapter 4 will further discuss trends in the area of graphical environments.

PICT is another example of a graphical environment that uses picture objects, called icons, to represent the files, programmes, facilities and actions that would traditionally be represented as words or mnemonics. [Gliner84]

These developments have given the user a wide choice in the type of interface that would suit his needs.

References

Barker85
Barker, P. and Yeates, H., "Introducing Computer Assisted Learning", Prentice Hall, 1985.

Bradle82
Bradley, J. "File and Data Base Techniques", HRW, University of Calgary, 1982.

Codd70
Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", Comm of the ACM, Vol 13, No 6, pp 377-387, June 1970.

COSMOS85
COSMOS, "Revelation User Manual Release G", COSMOS Incorporated, 1985.

Gliner84
Glinert, E.P. and Tanimoto, S.L., "Pict: An Interactive Graphical Programming Environment", IEEE Computer, pp 7-25, Nov 1984.

Held75
Held, G.D., Stonebraker, M.R., Wong, E. "INGRES - A relational database system", Proc. AFIPS, AFIPS Press, Vol 44, pp 409-616, 1975.

Hercul84
Hercules Computer Technology, "Hercules Graphics Card - Owner's Manual", Hercules Computer Technology, 2550 Ninth Street, Berkerly California 94710, 1984.

Hoye73
Hoye, R.E. and Wang, A.C., "Index to Computer Based Learning " , Educational Technology Publications, Englewood Cliffs, N.J. 07632, 1973.

Ingall81
Ingalls, D.H.H., "The Smalltalk Graphics Kernel", BYTE, pp 168-194, Vol 6, No 8, Aug 1981.

James81
James, E.R. "The User Interface: How we may compute" In Computing Skills and the User Interface, pp 337-371, Academic Press, 1981.

Kearsle82
Kearsley, G. "Authoring Systems in Computer Based Education", Comm. of the ACM, pp 429-437, Vol 7, July 1982.

Kernig76
Kernighan, B.W., "Software Tools", Addison Wesley, 1976.

Kernig79
Kernighan, B.W. and Lesk, M. "LEARN - Computer Aided Instruction on UNIX (Second Edition)" in UNIX Programmer's Manual Vol. 2, 7th Edition, Jan 1979.

Martin81
Martin, J. "Computer Data-Base Organization" Prentice Hall, 2nd Edition, 1977.

McDona74
McDonald, N. and Stonebraker, M. "CUPID the Friendly Query Language", University of Berkerley, California, Oct 1974.

Pike84
Pike, R. "The Blit, a multiplexed graphics terminal", in AT&T Technical Journal, pp 1607-1631, Vol 63, Oct 1984.

Relati84
Relational Technology Inc., "Introduction to INGRES", Relational Technology Inc., California, 1984.

Shirle78
Shirley, R., "Graphical Displays for CAL" in Interactive Computer Graphics in Science Teaching, John Wiley, pp31-54, 1978.

Snow78
Snow C.R., "The Software Tools Project", Technical Report Series , Univ. Of Newcastle Upon Tyne, No. 118, Jan 1978.

Starkw69
Starkweather, J.A. "A Common Language for a Variety of Conversational Programming Needs in Computer Assisted Instruction, pp 209-304, Academic Press, 1969.

Thomso75
Thomson, K. "UNIX IMPLEMENTATION", in UNIX Programmers' Manual, Bell Laboratories, Sixth edition, May 1975.

Martin81
Martin, J., "Computer Data-Base Organization" Prentice Hall, 2nd edition, 1977.

Zloof75
Zloof, M.M. "Query-By-Example: The Invocation and Definition of Tables and Forms", Proc. of the International Conf. on VLDB, pp 431-438, vol. 44, Sept 1975.

CHAPTER 3

Project - A Generalised Computer Aided Learning System

3.1 Requirements of a CAL system.

The earlier chapters have described the various approaches of CAL Systems and the tools concept. Some of the earlier CAL systems are powerful and impressive like the PLATO system, but have the problem of cost effectiveness. Latter CAL systems, like GUIDON, often utilise knowledge bases. GUIDON uses MYCIN. MYCIN was geared towards a specific area - medical diagnosis of infectious diseases. The advantage of GUIDON is in its expertise, on the other hand the system being specially designed for a specific purpose could not be easily adapted to teach a different subject.

Principal concerns in any CAL system design must be:

1) Cost-effectiveness - keeping costs down;
2) Effectiveness - ability to present subject material effectively;
3) Flexibility - ability to adapt to different subject matter without too much difficulty;
4) Portability - attractive for the system to run on a variety of machines.

Keeping in view these objectives, a project was undertaken to design a Computer Aided Learning System. This chapter describes the target users, their environment and the

CAL project.

3.1.1 Desired Behaviour

Any system is aimed at some particular class or classes of user. In a computer aided learning system the category of users can be identified as students, authors/writers of lessons and the administrator. It is assumed here that the functions of the administrator is taken on by the author/writers of the lessons. [Boulay81]

Users of a computer system will evaluate the system as a tool by virtue of the system's (tool's) ability to service his needs. How well the fit between tool and task will be the main criteria for assessment.

A second area is the expertise level required of the user. The user, whether student or writer, would wish to devote more of his time to his chosen field of study or interest. His knowledge of computer technology will likely be limited to the area of immediate concern. He will seek to minimize the time and effort he must devote to studying the intricacies of the system before he can be productive.

Ease of use is a third factor the user would require to minimize his time and effort in operating the system.

The fourth and final factor is the 'user support' aspect of the system. First time users need more help by virtue of their limited knowledge. They might face problems in understanding and interpreting how the system can handle these tasks, how to ask it to do so, why it is behaving as it is, and how to recover from errors and breakdown.

We could identify in our proposed CAL system two broad categories of users. The first type we call the Student User.

The student user, depending on his/her experience, may be further divided into one of these three groups:

'naive'	- with no programming/computing experience,
'novice'	- with some programming/computing experience,
'expert'	- with extensive knowledge of the system.

The second type of the proposed CAL system's user we called the Staff User. The Staff User would use the system in two ways, either as

'writer'	- author of the lessons for the CAL system, or
'maintainer'	- to update/enhance/correct the CAL system

programmes/files.

Having identified the different categories of users, we can proceed to determine the characteristics and needs of these users.

The Student User

The 'naive' student user will require the most assistance from the CAL environment and his progress would be much slower than other categories of users. The 'novice' will occasionally require some form of assistance and generally proceed at a quicker pace through the lessons of the CAL system. The 'expert' will perform the lessons quickly or merely browse through what is available in the CAL system. These different characteristics give us some indication of the needs:-

1) provision of help facility to aid the user on unfamiliar facilities;
2) user-friendly interface to provide hints on what to do next;
3) flexibility to avoid boring the 'faster' students.

The Staff User

The Staff User may perform the function of an author of the CAL lesson, or he might be maintaining the general aspects of the CAL system. In the first type of work, the CAL author will be involved in preparing, editing, testing and implementing the courseware lessons. The maintenance aspect will require monitoring of errors, updating/amending/enhancing the CAL programmes, documentation and files. The needs of the staff user are:-

1) an author language to facilitate the writing of lessons;
2) an editing/file handling environment to support the author in creation and editing of the lesson files;
3) some form of validation of the lessons written to minimise errors in the lessons before they can be implemented for use;
4) facilities to evaluate the performance of the courseware and student;
5) Facilities to allow the preparation/updating of the CAL system's programmes and documentation.

3.1.2 Software

An integral component of a computer system is the software, or the instructions, that the computer executes to support the various functions of its system. The basic level of software support in a computer system is the operating system. The operating system takes care of very low level housekeeping tasks like where a computer file is to be placed physically on a disc, checking if a device is busy, and so on. The operating system thus shields the computer user from many of the tedious and harsher aspects of the system. One level above the operating system, is the software which provides the facilities for the computer user to interact with the system. These facilities can fall into several groups:-

(1) FILE
(2) DEVICES
(3) PROGRAMMING LANGUAGES
(4) TEXT PROCESSING PROGRAMS
(5) EDITORS

The next level of software can be grouped as:- those written by the computer users. The software written by the users can well form part of the system's facilities thus blurring the different levels defined.

UNIX Operating System

UNIX operating system was developed at Bell Laboratories by Ken Thompson and Dennis Ritchie, the first edition was out in November 1971 implemented on a DIGITAL PDP 11/20 machine.[Thomso74]

UNIX was later developed to be a portable operating system when the original system on the PDP11 was moved to an Interdata 7/32 computer. Following the success of the initial experience in porting the system, a portable version VII (7) was developed which has been the starting point of moving UNIX to other machines. Portability of UNIX undoubtedly contributes to its popularity as an operating system. [Wallis82]

The success of UNIX, as a standard operating system, can be seen in the implementation of UNIX on different computers, ranging from microcomputers to main-frames, and its use in various establishments ranging from educational, government laboratories to the commercial and industrial. Many other systems have taken the philosophy of UNIX or MULTICS(the predecessor of UNIX), to provide a similar operating system environment. CROMIX, XENIX, TME are some of the operating systems developed from the same philosophy. The strength of UNIX lies in the flexibility of its programming development environment which has allowed many time consuming tasks to be eliminated.

The Seventh Edition of UNIX was made available on PDP11 16 bit computers and the UNIX 32V Edition on the VAX 11/700 series of computers from Digital Equipment Corporation (DEC). Some differences do exist between the different versions, both in the operating system and the commands. The UNIX 32V was later developed by the University of California at Berkerly and later versions for UNIX by Berkerly were known as BSD.

UNIX provides an attractive environment for programmers to work in, mainly because:-

(1) UNIX is a proven multi-user, multi-tasking environment;

(2) powerful and sophisticated range of programmers's tools are available;

(3) portability of software written on UNIX systems.

The factors mentioned above have influenced the author's decision in adopting UNIX as the software development environment for the CAL system. In the following sections, UNIX's software development tools to be used in the CAL system are discussed.

Editors

The editors commonly used on the UNIX system are 'ed' and 'vi'. 'ed' is more widely available because it uses only the basic facilities available on any terminal. 'vi' takes full advantage of video terminals. A screen editor called 'sc' is available on local UNIX machines and is widely used among the university community because of the support given by the local and regional (NUMAC - Northumbriam Universities Multi Access Computer) computing centres and its implementations on a wide number of machines under different operating environments. 'sc ' is an attractive editor to learn as it has been implemented on a variety of machines in the University. Users need not learn how to use the features of a new editor when using different machines since 'sc' is available on most if not all machines in the university.[Joy80] [Hunter82]

Editors are general purpose tools meant for creating or editing a document in the computer. The document can be just plain text or source programmes. Older file editors assume that the typical user is from the scientific community, spending most of their time editing programmes and preparing the selected data files. Users now are mushrooming from other areas, like text preparation and formatting. Hence the demands on editors are greater in terms of facilities provided. Many features of word processing are incorporated in screen editors to provide e.g. indentation, automatic line insertion, tabulation and the like.

In our requirements - editors will be used in operating the lesson text. A special purpose editor would be ideal. Editors are generally and inherently very large programmes that need to be robust and reliable. Thus rather than writing a customised editor a good compromise would be to adapt an existing editor to suit the needs of the CAL system.

Programming Languages

Choosing the programming language for the system can be difficult when there are a number to choose from the installation's UNIX environment. Two programming languages that were considered are PASCAL and C. [Elfrin85]

PASCAL is a high level programming language that has the advantage of portability (if written in standard PASCAL) and strong type checking. Though C, like PASCAL, is a high level language, and has type structures similar to PASCAL, it has less strict type checking. C has the advantage of flexibility and efficiency.[Wirth75] [Kernig78]

Portability was mentioned as a desirable aspect of any software. Portable software are used in the sense that the software could be transported to different computers with less effort than would be needed to redevelop the software for the second computer. Programmes that can be used on other machines means lower costs in development and implementation.

UNIX being a portable system itself, was written mainly in the C programming language. Compatibility and portability are the main factors that made the author decide to develop the CAL software in C.

Storage

UNIX organises its directories and files in a hierarchical or treelike structure. A file on UNIX is basically a one-dimensional array of bytes, the files are attached to a hierarchy of directories. Directories themselves are files that users cannot write into.

An alternative to storing information on conventional UNIX files is to have it on a database system. There is a set of database library functions on UNIX under the name of 'dbm' that could manage and maintain a simple database system. The library functions can be called from a C programme to create a database and perform retrieval, updating and other database functions on a database. The resulting database created using 'dbm' consists of 2 UNIX files with names suffixed by the following '.dir' and '.pag'. The '.dir' file is a bit mapped file and is used by 'dbm' to indicate if a record is present in the '.pag' file. The '.pag' file contains all the records in the database. A hashing routine is used to place the records and fetch it from the database. Records in the database have to be within a logical block size, in the case of the PDP 11/44 it is 512 bytes per block. On the VAX 11/750 the block size varies depending on how the disc partitions are logically blocked. Figure 3.0 shows that block sizes are at 8192 bytes for files under the partitions **ba**, **bd** which are directories mounted on /(root directory) and **/user/staffl.** This means that 'dbm' functions on the VAX could be set to work on records with 8192 bytes if the database were to reside on the partitions logically blocked at 8192 bytes e.g. / and **/user/staffl.** Each record in the 'dbm' database consists of a key part and a content part.

The advantages of organising data using 'dbm' are:-

(1) very large database of over a billion blocks can be handled;

(2) access time is fast, a fetch takes between 1 to 2 system accesses;

(3) it is simple and easily incorporated into an application programme.

Figure 3.0   Disk partitions on the UNIX BSD4.1 (VAX 11/750)
             (at dept. of Computer Science, Durham University)

```
%df

Filesystem    kbytes    used    avail  capacity   Mounted on
/dev/up0a       7413    4900    1771     73%      /
/dev/up0f      66415   58492    1281     98%      /usr
/dev/up0e      26235   24962       0    106%      /usr/src
/dev/up1d       7413    5283    1388     79%      /user/staff1
/dev/up1e      26235   22382    1229     95%      /user/staff
/dev/up1f      66415   50554    9219     85%      /user/student


% cat /etc/disktab

# disktab 4.5 83/07/30
# Disk geometry and partition layout tables.
# Key:
# ty type of disk
# ns #sectors/track
# nt #tracks/cylinder
# nc #cylinders/disk
# p[a-h] partition sizes in sectors
# b[a-h] partition block sizes in bytes
# f[a-h] partition fragment sizes in bytes
#
# All partition sizes contain space for bad sector tables unless
# the device drivers fail to support this.

#
# Disks normally on up
#
160|fuji|fuji160|Fujitsu 160:\
 :ty=winchester:ns#32:nt#10:nc#823:\
 :pa#15884:ba#8192:fa#1024:\
 :pb#33440:pc#263360:\
 :pd#15884:bd#8192:fd#1024:\
 :pe#55936:be#4096:fe#512:\
 :pf#141600:bf#4096:ff#1024:\
 :pg#213600:bg#4096:fg#512:
```

pa is the swap disk partition
pc is the whole disk(not used)

UNIX LEARN

UNIX's LEARN was conceived by its authors (Brian Kernighan and Mike Lesk) to be the main computer aided instruction medium for the novice user of the UNIX Operating System. The novice user needs to know some basic operations on the UNIX system before he could use LEARN, a short presentation and orientation by a demonstrator could achieved this preliminary task. Once the user has logged into the UNIX system, LEARN could then be the means of acquiring basic skills in using UNIX. By typing "learn" at the system prompt (then the RETURN key) the LEARN system would then be in control till the user decides to end the session by executing the LEARN command "bye". During a LEARN session the user would be presented with the relevant materials (or scripts) on the chosen topic. Each presentation of a frame (usually a screenful) of material will require some response by the user to test the user's comprehension of the materials presented so far. The user's response is checked against one or more of the expected responses. If the response is correct the next lesson material will be presented, if the response is incorrect then the standard response would ask the user if he wished to retry the previous question. Flexibility in catering for partially correct responses could be provided for by the author of the lessons to give hints to the user in that eventuality. Users could at the start of LEARN choose the topic as well as the lesson number to go to thus avoiding repetition of going through completed lessons. LEARN provides a log of the user's performance and could if provided by the lesson author skip certain lessons depending on the speed and performance of the user session, to avoid boring the faster students.[Kernig79]

LEARN (Second Edition) originally covered 6 topics on the UNIX system, they are :-

(1) files - teaching basic file handling techniques on the UNIX system;

(2) editor - teaches about the UNIX text editor 'ed';

(3) morefiles - more advanced file manipulation and commands;

(4) macros - teaches about use of 'ms' macros for the 'nroff/troff' document preparation commands;

(5) eqn - shows how mathematical typing can be done;

(6) C - introduce writing of programmes in C.

Students at Durham University on their first few computing practicals on the UNIX systems will have to go through the first two or three topics of LEARN before they proceed onto the actual programming exercises. Students using LEARN during their first few sessions would find it more helpful if LEARN could provide some features that could guide the novice user on certain LEARN commands that are not explicit.

Students encountering a long lesson script, especially if the lesson text is greater than can be displayed on a screen, may wish to reread the text. This is not possible in the version of LEARN (2nd edition) unless the student exits from the LEARN session and reenters LEARN, specifying the subject and lesson number he wished to go to. There is no facilty in LEARN to allow the student to view how he has performed so far, though a performance log file can be maintained by LEARN for this purpose.

The original LEARN programme consists of 15 seperate modules, with about 1500 lines of C code in all. This LEARN programme could form the basis of the teaching system since it has been a tried and tested system. Modifications can be incorporated to provide a 'friendly' interface and test the use of retrieving lesson scripts from a database(dbm).

3.1.3 The User Interface

The key notion behind a user interface is that the user and the computer are engaged in a communicative dialogue whose object is to accomplish some task. Dialogue is used because both computer and user have access to the stream of symbols that flows back and forth to accomplish the communication.

The typical physical user interface a few years ago was the teletypewriter, currently it is the alphanumeric (keyboard) video terminal.

In the local installation, Televideo TVI 910 and 912 alphanumeric video terminals are used as the main input and display devices. The UNIX system supports a command-line interface. This form of interaction can be difficult for a novice user because UNIX commands are terse and sometimes cryptic e.g.

to list the files in a directory the command is "ls"

Windowing Software

There is available on UNIX BSD4.1 a software library package written by Ken Arnold called "curses". The package has a set of functions which could be called to perform cursor addressing, creation of windows and a host of other screen updating functions. These screen updating functions rely on a terminal capability database known as 'termcap' which describes the capabilities of the terminal.[Arnold80]

The availability of the "curses" package alleviates the difficulties involved in creating a different interface from the command-line type. A menu type interface could be created and the maintenance of the screen interface with respect to the terminal types can be left to the 'termcap' database and its related utilities.

HELP

A help facility to aid the user will be added. Most systems in general offer some form of on-line help facility which assist the user by explaining command words and giving examples. The aims are:-

1) provide an on-line documentation;
2) a friendlier user interface;
3) better understanding of the system.

HELP on UNIX consists of documentation on the usage of commands invoked by the "man" (for manual) facility. This is quite unsatisfactory for a novice user as he will not be in a position to know what commands exist in the first place.

A help facility should be easily activated. Conventionally most are activated through typing the word "help". This would normally lead to a display of a summary of what to do next.

UNIX's "man" command prints out the manual section for a command. Typing "help" in UNIX (4.1 BSD) will list some common commands with a short description of what the command does and recommends the use of the 'man' command to find out more about UNIX commands. This is not adequate for a novice user who is finding his way around the system because the help text does not display most of the commands, and the user would need to know what the command name is before he could find out more using 'man'.

In a CAL system, the users would benefit from a friendlier environment to proceed through the system and to achieve this a more comprehensive help facility and a windowing based interface are required.

3.1.4 Summary

The facilities and software available in the UNIX environment form the basic tools for creating a computer aided learning system. From the requirements and issues discussed earlier the following major components of a CAL system are identified:-

1)	TEACHING system	-	existing LEARN system with modifications in C.
2)	WRITER system	-	existing screen editor 'sc' with modifications in C. A prototype environment to create, test and install the lessons.
3)	USER INTERFACE	-	menu and command line interface written in C. The 'curses' library package used to facilitate the creation and updating of the screen interface.
4)	DATA BASE	-	lesson scripts will be stored on a database using the 'dbm' database library package and the retrieval/ update programmes written in C.

5) HELP - use of the UNIX "man" command and software written in C.

The idea is to rely on the available software wherever possible and adapt the software to form an integrated CAL system. The five major component parts of the CAL system will be described in more detail in the design and testing cycle.

3.2 System Design and Implementation

This section describes the final design of a computer aided learning system implemented in a UNIX environment. Hence the system is given the name CALUNIX for Computer Aided Learning on UNIX.[Jackso83]

3.2.1 Overview

A top-down view of the various CALUNIX components is shown in the following diagram:-

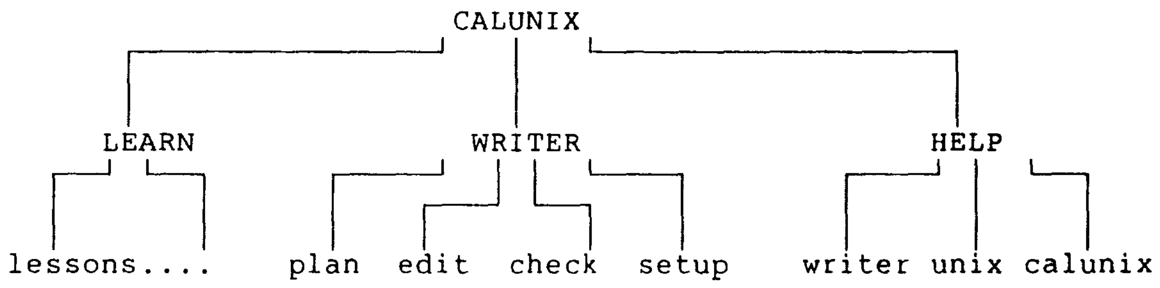


Figure 3.1 CALUNIX Overall System Structure.

The main CALUNIX programme would control the lower level modules from user's input using a menu (shown in Figure 3.1) driven format.

Program Documentation for CALUNIX

NAME

CALUNIX - Computer Aided Learning on UNIX

SYNOPSIS

```
CALUNIX [ option [ subject / Help Option ] *lesson] *speed]]
```

DESCRIPTION

CALUNIX provide an environment to learn and write lessons on the UNIX system. To get started type 'CALUNIX'. A menu with the options will be presented and the programme will prompt for an option. The options presently handled are:

```
learn
writer
help
```

To skip the menu and proceed straight to the option type in the option in the command line (% is the prompt) example:-

```
%CALUNIX learn pascal
```

or

```
%CALUNIX writer pascal plan
```

To exit enter the BREAK key or the break key sequence ("CTRL & C" keys on Televideo terminals).

The options *lesson and *speed only work when the LEARN option is specified as the second parameter in the command line.

3.2.2 The CALUNIX LEARN Component

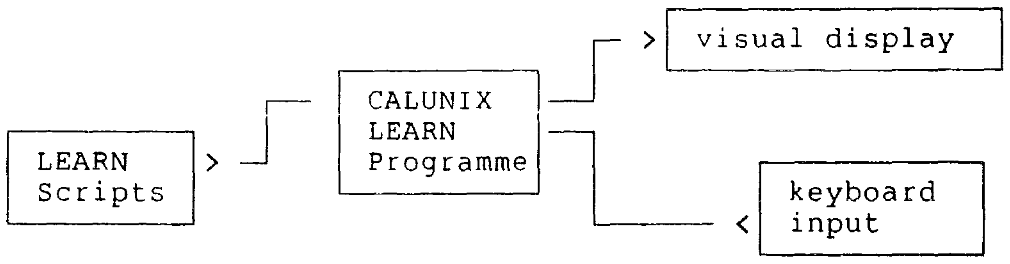


Figure 3.2 CALUNIX LEARN interaction.

The LEARN component retrieves the lessons from either a UNIX file or records in a database(dbm) :-

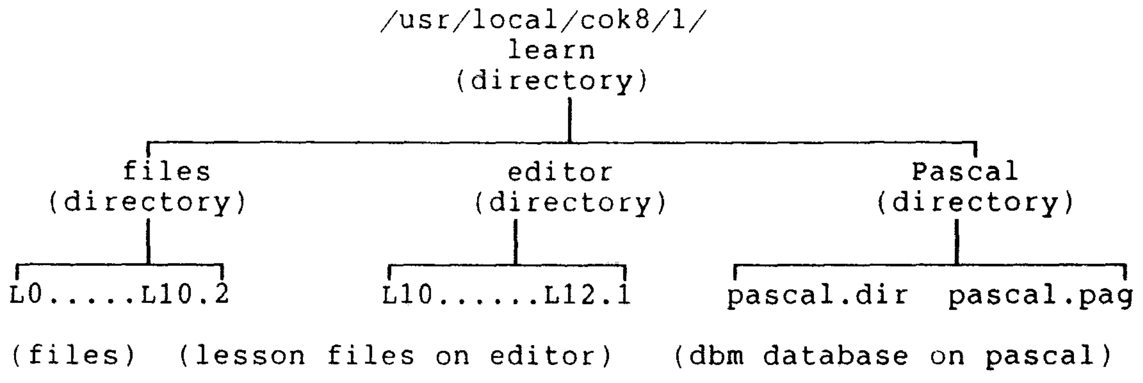


Figure 3.3 Directory/File structure of LEARN lessons

CALUNIX LEARN, in addition to retrieving the lesson scripts, be it in a database or as a UNIX file, has also to:-

1) interpret the lesson scripts;
2) control the learning session;
3) provide certain user controls to obtain help or control the displays.

The LEARN component is basically the same as the UNIX learn.

Program Documentation for:-

NAME

(CALUNIX)LEARN - modified LEARN - Computer Aided Instruction
about UNIX.

SYNOPSIS:

CALUNIX LEARN

DESCRIPTION

(CALUNIX) LEARN is similar to the LEARN on UNIX. It is called from CALUNIX. The LEARN programme will ask questions to find out what you want to do. Some questions may be bypassed by naming a subject and also including the lesson name. The current (CALUNIX) LEARN only handles lessons on the subject 'pascal' stored in a database.

A sample session on the CALUNIX LEARN option is shown in the following pages(see Figure 3.4). The session was recorded in a file using the UNIX command 'script' then going through the CALUNIX menus and questions.

Figure 3.4 SAMPLE SESSION ON CALUNIX LEARN

```
Script started on Wed Sep  3 14:56:11 1986
$ CALUNIX
```

```
+--------------------------------------------------------------------+
|                      CALUNIX    MENU                               |
|                                                                    |
|   CALUNIX is a Computer Aided Learning environment on UNIX         |
|                                                                    |
|   *** The following options are available on CALUNIX:-             |
|                                                                    |
|                                                                    |
|       (1) LEARN option  - to learn about a subject;                |
|       (2) WRITER option - to write lessons on a subject;           |
|       (3) HELP option   - to give help on CALUNIX options;         |
|                                                                    |
|                                                                    |
|    Hit the BREAK key for option to exit from  CALUNIX.             |
+--------------------------------------------------------------------+
|                                                                    |
| Which Choice(Option)?learn                                         |
+--------------------------------------------------------------------+
```

```
These are the available courses -
  files          )
  editor         )
  morefiles      )   NOT AVAILABLE
  macros         )   ON THIS
  eqn            )   TRIAL YET!
  C              )

  pascal             available
If you want more information about the courses,
or if you have never used 'learn' before,
type 'return'; otherwise type the name of
the course you want, followed by 'return'.

files - basic file handling commands
editor - text editor; must know about files first.
morefiles - more on file manipulations and other useful stuff
macros - "-ms" macros for BTL memos & papers;  must know editor
```

Figure 3.4 cont.

```
eqn - typing mathematics;  must know editor
C - writing programs in the C language;  must know editor
pascal  -  writing programs in the Pascal  language;  must  know
editor

This is probably the proper order, but after you
have the "files" course and know the basics of "editor",
try anything you like.

You can always leave learn by typing "bye" (and a RETURN).
You can stop it from typing by pushing interrupt
(or break or rubout or delete, depending on your terminal).

If it won't accept your answer, and you know you're
right, answer "no" when it asks whether you
want to try again, and it will go on to the next lesson.

Hit spacebar then RETURN key to continue
Hit the B key & then RETURN to  display last 20 lines
You can 'mail'  any problems concerning with use of  learnpascal
to scnk.

Which subject?
Which subject?  pascal
If you were in the middle of this subject
and want to start where you left off, type
the last lesson number the computer printed.
To start at the beginning, just hit return.

      You have signed on to the Learnpascal program.
      But before starting you should first be able to
      correct typing mistakes at the terminal using
      the special characters (left arrow) and CONTROL-U.

      (left arrow) cancels the previous character typed;
      (It  is  made  by pressing the key marked  with  an  arrow
pointing left)
      CONTROL-U cancels the line being typed;
       (It is made by holding down the key marked CTRL
        or CONTROL and pressing the U key.)
      If you make a typing mistake, you can use these
      characters to correct it before you finish the line
      and the computer will never know about it. For example
      what will the computer receive if you type

      st(left arrow) he(CONTROL-U) thf(left arrow)e

      The symbol (left arrow) shows where the left
      arrow key was pressed.
      Similarly for the CONTROL-U function.
```

Figure 3.4 cont.

```
Hit spacebar then RETURN key to continue
Hit the B key & then RETURN to  display last 20 lines

      Reply 'answer Word' where Word is the word as it will
      be received. For example if you think it will get 'dog'
      then type
               answer dog
      Dont forget to leave a space between answer and the word
      and to hit return at the end of the line. Also dont use
      quotation marks in your answer.
$ $ answer the
 very good !!!!!

Good.  Lesson 0.1a (1)

      Some of the lessons will be longer than one terminal
      screen height. The computer will always go on until
      just leaving the last part visible. So it is useful
      to freeze the screen when it is full this can be done
      using the CONTROL-S for stopping and then the CONTROL-Q
      for restarting.

      Firstly there is a section on the history of Pascal
      Type in answer start to begin.
$ answer start

Good.  Lesson 0.1b (2)

      The computer language PASCAL was the first language to
      embody in a coherent way the concepts of structured
      programming defined by Edsgar Dijkstra and C.A.R.Hoare.
      PASCAL was developed by Niklaus Wirth in Zurich, it is
      derived from the language ALGOL 60 but it is more powerful
      and easier to use. PASCAL is now widely accepted as a
      useful language particularly as a teaching tool.

      Useful books are
        1 Pascal user manual and report by Jensen and Wirth
        2 Programming in Pascal by Grogono
        3 Problem solving using Pascal by Skvarcius

      Type in answer ok  if you are ready
$ answer ok

Good.  Lesson 0.1c (3)

      This is a very simple Pascal program which
      is used to show what the structure of a
      program is like. All the programs shown are
      complete working programs that may be run
```

Figure 3.4 cont.

on a computer.

```
PROGRAM squarerootoftwo (output);
   BEGIN
      write (sqrt(2))
   END
```

This would print 1.4142135624 which is approx the square-root of two. Even a simple program must obey certain rules shown in the program heading syntax diagram. A syntax diagram shows the rules of a particular construction.

Figure 3.4 cont.

```
      program---->[heading]----->[block]----(.)----->

      heading---->(PROGRAM)--->[identifier]------
Hit spacebar then RETURN key to continue
Hit the B key & then RETURN to  display last 20 lines
                                                 |
      ---------------------------------------    |
      |
      |-->(()--->[identifier]-----())----->(;)------->
              |                 |
              |---(,)---------<-|

      What is the first word of any Pascal program?
      Type in answer X where X is the word.
$ $ answer wrong           (Deliberate wrong answer given)
```

```
                    CALUNIX  LEARN

     Sorry the answer was not right  !

      CHOOSE ONE OF THE FOLLOWING OPTIONS HERE :-
         (1) Try again - display lesson again------1
         (2) Try again - don't display text--------2
         (3) Skip lesson---------------------------3
         (4) help/view - help or view lessons------4
         (5) bye - exit from learn-----------------5

  OPTION?4
```

```
                    CALUNIX  HELP

     CALUNIX HELP provides the following options


         (1) LEARN  option - Help about learn;
         (2) PLAN   option - View lesson plan;
         (3) LOG    option - View performance log;


       Hit the  BREAK  key for option to exit from  CALUNIX

   Which Choice(Option)?
```

Figure 3.4 cont.

(The BREAK key was activated at this point)

```
Interrupt.
Type h if help required, y if you want to go on?n
Bye.
```

```
                   CALUNIX    MENU

   CALUNIX is a Computer Aided Learning environment on UNIX

   *** The following options are available on CALUNIX:-


       (1) LEARN option  - to learn about a subject;
       (2) WRITER option - to write lessons on a subject;
       (3) HELP option   - to give help on CALUNIX options;


    Hit the BREAK key for option to exit from  CALUNIX.

 Which Choice(Option)?
```

(The BREAK key was activated again at this point)

```
 MAIN CALUNIX ROUTINE--Interrupt.
Want to go on? n
$
$
script done on Wed Sep  3 15:02:02 1986
```

END OF Figure 3.4

3.2.3 The WRITER/AUTHOR Component

The writer/author component aims to provide an authoring system environment for the lesson writer. The facilities include a screen editor, lesson checker and lesson setup programmes. A synopsis showing the structure of the writer component modules in relation to the CALUNIX system is as in Figure 3.5 below:-

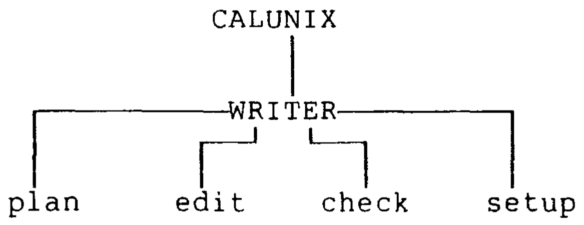


Figure 3.5 CALUNIX WRITER calling structure

1) The Plan module calls the screen editor for the writer to build up the contents description on a subject.
2) The Edit module uses the same screen editor but this time for the writer to build/edit the lessons.
3) The Check module validates the lesson created.
4) The Setup module will attempt to set up the lesson files in a database.

In the following pages a description of the displays for the writer component is presented.

The lesson writer has first to select the subject he will be working on, then he will be prompted to select one of the four writer options available. The sequence of options follows a top-down approach. The writer is encouraged to plan the lesson structure first, via the plan option, before proceeding to the edit option to create and edit the lesson proper. After a lesson is created the check option can be called to validate the system of the author language commands used.

On completion of all the lessons, the writer can call upon the setup option to set up the lessons in a database. The database set up has the advantage here of a fast system access to the lesson files and a more secure environment for student users.

Programme documentation for:-

NAME

WRITER - An authoring system for CALUNIX LEARN lessons.

SYNOPSIS

WRITER [subject [option [lesson]]]

DESCRIPTION

WRITER provides an environment to plan, write, check and setup lessons on a subject. The lessons could then be used by students through learn(see CALUNIX LEARN)

To get started simply type 'WRITER'. (Remember it must be in capital letters). The programme will ask questions to find out what you want to do. The questions may be bypassed by naming first a subject, and then an option, chosen from the following:-

plan	-	to write the lesson plan,
edit	-	to edit lessons,

```
check -  validate lessons,
setup -  setup lessons in a database,
```

plan option is an aid to writer in organising the lesson structure and description for later reference, the screen editor (see sc) is used and the header is placed as follows to encourage a prescribed format of entry:-

```
====.====|LESSON NO|====.====|LESSON DESCRIPTION====.
             1                  Introduction to pascal
             2                  History of pascal
             3                  A simple pascal programme(ex)

            ...                 .........
```

edit will prompt for the lesson name to be given, the name is checked; an existing name will result in editing of the contents of the lesson file named, while a new name will result in the creation of a new empty lesson file for editing. The screen editor is used. When in screen editor mode, help on the learn author language can be obtained by hitting ESC key twice and selecting the appropriate help required.

check will prompt for a lesson name, the lesson file named will be checked to minimise any errorneous lessons for the student user when he is using learn on the lessons.

setup is to be used only if the lesson is to be set up on a database (see dbm). The only advantage of setting up a lesson on a database is a faster retrieval of lesson files, provided all or most of the lesson files are within 1 block in size (PDP11 - 512 characters). It should be performed if at all when all lessons files have been completed and checked.

The key BREAK is used to terminate a WRITER session.

The WRITER menus and screen displays are described here:-

```
                    CALUNIX WRITER

CALUNIX  WRITER supports the lesson writer with the  following
set of tools.


(1)  First you must specify the subject you are writing on.

(2)  Then choose one of the following options:-

     a)   PLAN option   - to plan lessons on subject;
     b)   EDIT option   - to create and edit a lesson;
     c)   CHECK option  - to check/validate lesson;
     d)   SETUP option  - to setup the lesson;

(3)  For option 2b),  c) and d) you must give the lesson name/
number

-------HIT-BREAK-KEY-FOR-OPTION-TO-EXIT-----------------------


     which subject? pascal  (RETURN)

     which choice (option)/ Plan  (RETURN)
```

Figure 3.6 CALUNIX WRITER menu

Choosing the subject pascal and the option plan will cause WRITER to call upon the screen editor to create and edit a file called planpascal (concatenating option plan with subject pascal) shown in Figure 3.7.

```
planpascal                line:1                   column:1
====.====|LESSON NO|====.====|LESSON  DESCRIPTION|====.====|

              0             Start

             0.1a           Characteristics of Televideo
                            TV1912
             0.1b           Terminal TV1912 (continued)
             0.1c           Introduction to Pascal/Books
             1.0            A simple Pascal Program
             1.1            Pascal Program Name Exercise 1
             1.1a           Pascal Program Name Exercise II
             2.0            Pascal Operators: Assignment I
             2.0a           Statement Operator Exercise I
             2.1            Research Words-word delimiters I
             3.0            Identifiers I
             3.0a           Identifier Exercise II
             3.1            Identifier Exercise III
             3.1a           Full Identifier list
             4.0            Literals and Constants ExI
             4.0a           Literals and Constants Ex II
             5.0            Pascal Notation and Vocabulary
             5.1            Pascal INTEGER Type Ex I
             5.1a           INTEGER Operators Ex II
             5.1b           Hierarchy of Operators Ex III

====.====|====.====|====.====|====.====|====.====|====.====|
```

Figure 3.7 Sample of lesson plan for PASCAL using 'sc'.

The screen editor used is a version of the NUMAC 'sc', the difference being only in the screen template displayed and the additional help text (on the author language) provided under the 'Writer Commands' options.

The EDIT option screen editor template is as shown in Figure 3.8. In this Figure 3.8 a lesson 1.0 (LI.0) is created and edited.

```
L1.0          line:1                    column:1
====.====|====.====|====.====|Writer HELP - Types ESC twice=.====









====.====|====.====|====.====|====.====|====.====|====.====|====.==
[press ESCAPE twice for STOP or HELP functions
```

Figure 3.8 Screen editor 'sc' template for WRITER EDIT

In the screen editor typing the ESC key twice will display the following prompt in the lower ruler line window:-

```
====.====|====.====|====.====|====.====|====.====|====.====|== ==.=
Stop, Help, Position, Option, Writer Commands...
```

Figure 3.9 Options in WRITER EDIT screen editor

Any one of the five options can be selected by typing the first letter of the option word and pressing the RETURN key. e.g. if 'H' (Help) was chosen, the help options will overide the previous options displayed in the lower window:-

```
====.====|====.====|====.====|====.====|====.====|====.====|== ==.=
Cursor, Action, Function, Block, Position, Option, Setup or Writer
```

Figure 3.10 Help options in WRITER EDIT screen editor

In the same way as the previous step choosing say Help on WRITER by typing W and the 'RETURN' key will display on the screen the following Figure 3.11:-

```
Writer: Short description of learn Author language commands

#print: display text that follows, up to a line that starts with #.
#create filename: creates a file of the specified filename.
#user: gives control to the user at this point, pass to shell for
       execution.
#copyin: #uncopyin: anything typed by the user between these
       commands are- copied to a file called .copy.
#copyout: #uncopyout: between these commands any material typed
       at the user by any program is copied to a file called .copy.
#pipe; #unpipe: to allow material typed between these commands to
       be fed - through a pipe so that sequences fed on editor
       such as ed will work.
#cmp file1 file2: compares the two files for identity.
#match stuff: the last line of the user input is compared to stuff,
       the success or fail status will be set according to it.
#bad stuff: same as #match except that it corresponds to specific
       wrong answers.
#succeed, #fail: will print a message upon success or failure.
#log: performance is logged - date, lesson, user id, speed rating,
       status.
#next: is followed by the next few lines each with a successor
       lesson name and an optional speed rating - as shown here:-

----------------------25.1a       10

Enter another HELP keyword or press RETURN to restore file image.
Cursor, Action, Function, Block, Position, Option, Setup or Writer.
```

Figure 3.11 Help text on WRITER commands in 'sc'

If more detail is required of the writer commands, selecting the 'Writer Commands' (Type W and 'RETURN' key) will display in the lower ruler line window:-

```
====.====|====.====|====.====|====.====|====.====|====.====|====.====|=
Summary,Copyin Copyout,Match Bad Fail Succeed,Next Log Print,Pipe User
 Cmp..
```

Figure 3.12 Additional help options for WRITER commands.

The 'Summary' option if selected by typing 'S' displays:-(Figure 3.13)

Figure 3.13 Summary of Writer Commands for 'learn' lesson text

```
        The Lesson Writer will need to know the 'learn' commands,
which  forms part of the lesson text instructing  the  'learn'
interpreter  on  what to do next during  the  CALUNIX  'learn'
session.

        All  learn commands begins with a # sign,  the  following
blocks  of  options  following this  summary  describes  these
commands in more detail:-

Copyin&Copyout      -     describes #copyin, #uncopyin,
                          #copyout, #uncopyout;
Match&Bad&Succeed -     describes #match#bad, #fail,
                          #succeed;
Next&Log&Print      -     describes #next, #log, #print;
Pipe&User&Cmp       -     describes #pipe, #unpipe, #user;

        TYPE IN THE FIRST LETTER OF THE BLOCK FOR ANY COMMAND
-----SHOWN IN THE BLOCK      AND      HIT THE RETURN KEY------

Enter  author Writer HELP keyword or press RETURN  to  restore
file image
```
Summary, Copyin Copyout, Match Bad Fail Succeed, Next Log Print, Pipe User Cmp.

A more detailed explanation of the commands say, about "#copyin", can be initiated by typing the first letter among the grouping of words. Say "C" is typed, the resulting help text is shown in Figure 3.14.

Writer's #copy #copyout commands

#copyin & #uncopyin

Anything typed between these two commands are copied to a file called .copy. This lets the other writer (learn) commands interrogate the student's (user) response upon regaining control from #user

#copyout 7 #uncopyout

Any material typed at the student by any program is copied to a file called .copy. This lets lesson writer check the effect of what the student typed, which true believers in the performance theory of learning usually prefer to the student's actual input.

Enter another Writer HELP keyword or press RETURN to restore file image.
Summary, Copyin Copyout, Match Bad Fail Succeed, Next Log Print, Pipe User Cmp..

Figure 3.14 More detailed help text for WRITER commands

Figure 3.14 shows the most detail help text available on the WRITER commands for the LEARN lesson scripts. Once the writer exits from the screen editor the CALUNIX WRITER menu will reappear and prompt for input.

The EDIT option

Choosing the EDIT option requires the sequence of entries from the user as follows:-

```
     subject?
     option?

     Lesson   Number/name/
```

Once the entries have been accepted, the EDIT module creates/edits a file with the name given in lesson Number/Name entry using the screen editor (sc). The screen editor top ruler template is different from the top ruler template of the PLAN option's screen editor. If the Lesson Number is 0 say the top ruler template would display:-

```
L0          line 1                column 1
====.====|====.====|====.====|Writer HELP - Type ESC twice=|====.
```

The 'Writer HELP - Type ESC twice' banner is to indicate to the writer how help on the author language commands can be initiated.

The CHECK option

CHECK option will ask for a lesson Number/Name to validate on. If the lesson is found CHECK will perform some simple checks on the valid author language commands.

The SETUP option selected will display the following message:-

```
THIS OPTION setup SHOULD ONLY BE USED WHEN YOU HAVE:-

     1)  created the necessary lessons;
     2)  you wish to set it up on a database (dbase):

THE LESSONS WOULD WORK JUST AS WELL IF IT WERE LEFT AS IT IS
Want to go on?
```

These options together form the authoring system provided under CALUNIX WRITER component.

3.2.4 THE HELP ENVIRONMENT

Overview of general help on CALUNIX:-

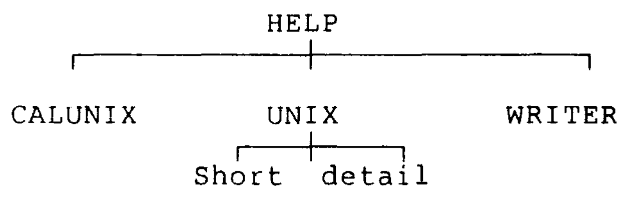


Figure 3.15 Overview of CALUNIX HELP

Help provides a description on CALUNIX and WRITER programmes through a system call of the UNIX 'man' command. Help on UNIX itself is a listing of the UNIX commands available, and a facility to provide a short or detailed description on each of the UNIX commands. (See Figure 3.17)

Help programme is written in C, the module handling the 'UNIX' commands uses library functions that maintain key/content pairs in a database. This library of database subroutines are described under 'dbm' in the manual pages ('man').

The screen interface for the UNIX option is divided into two windows shown in Figure 3.16.

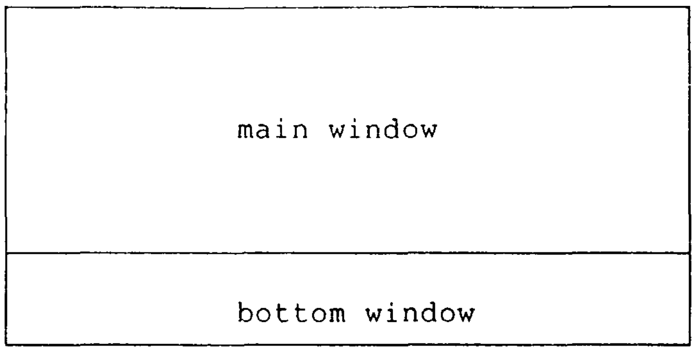


Figure 3.16 HELP Window Screen Interface

The main window displays the text while the bottom window displays the dialogue and short messages.

Figure 3.17 shows the actual help text in the main window and the dialogue text in the bottom window. The intensity of the text in the main window is lower to provide a highlighting effect between main and sub windows.

```
signal  chdir   cu      who     ecvt    creat   tsort   test
profil  rm      time    13tol   tp      yacc    putc    mkfs
quot    mktemp  sort    fread   su      mail    plot    ld
grep    xsend   lock    chown   dc      utime   atof    cp
end     pause   date    prof    write   sleep   mv      cc
nm      mount   cat     join    scanf   crypt   malloc  struct
pkopen  lseek   abs     ps      find    sync    chmod   touch
mkdir   echo    as      mp      exp     stat    kill    tabs
gets    rev     ac      ed      dup     read    od      mesg
acct    sh      getpid  assert  unlink  make    string  sed
out     ls      times   fopen   true    sin     close   pwd
dcheck  setbuf  pipe    rand    brk     system  wall    f77
stdio   printf  exit    look    wc      perror  link    pr
tty     mkconf  diff    tee     at      newgrp  file    icheck
man     pstat   bas     ptx     spline  deroff  tc      tetuid
col     size    in      cmp     fclose  stty    bc      umask
setjmp  clri    lorder  access  lint    exec    setuid  indir
ratfor  tbl     qsort   units   dd      du      ncheck  diff3
iostat  sum     nlist   hypot   refer   restor  strip   uucp
getenv  cal     tail    ctype   ctime   factor  split   uniqroff
Type y to continue/type in COMMAND for description ? who

who \- who is on the system who (1)
```

Figure 3.17 Help on UNIX commands

HELP in LEARN mode.

Two methods of initiating help when in LEARN can either be by an explicit call or by some condition being encountered. The same screen-interface of a main text window showing the help options available, and a bottom sub-window for the dialogue, is used when the help in LEARN is initiated.

HELP in WRITER mode

Help in WRITER mode is incorporated into the screen editor. Help here describes the CALUNIX WRITER author language commands, as described in the previous section 3.2.4.

3.3 DEVELOPMENT

This section outlines the different stages in the development of the project and describes the problems encountered during the project.

3.3.1 CONSTRAINTS AND PROBLEMS ENCOUNTERED

The help facility was the first set of programmes to be developed on a VAX, with the help of the 'curses' library functions to create the window interface, and the 'dbm' library functions to manage the help text in a key/content database.

Porting the help programme over to a PDP 11/44 also meant that 'curses' library functions had to be ported over since the 'curses' is available only on UNIX BSD 4.1 UNIX (VAX) but not on the UNIX VII (PDP 11/44). 'Curses' library functions depend on the 'termcap' database. The 'termcap' database describes the terminal capability of a number of terminal types. To set up the 'termcap' database the 'man' documentation stated the need for the new 'tty' drivers to be available. 'tty' is the general terminal interface. The problem arose when it was discovered that the UNIX VII (PDP 11/44) did not support the new 'tty'. Thus an alternative method has to be found to perform the windowing and cursor addressing.

The installation uses Televideo 912 and 910 terminals. A subroutine in help was added to support some basic cursor addressing and visual attributes for the Televideo terminals to make up for the scrapping of the 'curses' library functions[Televi82].

Problems encountered with 'dbm'

The 'dbm' library functions allow key/content pairs in a database to be accessed very quickly. However one limitation is that the sum of a key/content pair must not exceed the internal (logical) block size of the disk. In the case of UNIX VII on a PDP 11/44 the internal block size is 512 characters per block. In UNIX BSD 4.1 the block size can be 1024, 4096, 8192 characters per block depending on which disk partition the database resides in, and also the block size defined when compiling the 'dbm' library routines. The internal block sizes for the different disk partitions are described in the 'disktab' file on UNIX BSD 4.1 /etc/disktab. Figure 3.0 in Chapter 3 gives the disc partitions description on the VAX's disc storage unit.

Though the original authors of LEARN have recommended breaking down long lesson scripts into smaller ones, many of the lesson scripts would still exceed the 512 characters. This meant that a lesson script greater or near 512 characters in size could not be stored as a single key/content record in a 'dbm' database. Thus the advantage of optimal system access using 'dbm' would be difficult to be realised for UNIX VII. An added complication if 'dbm' is to be used is for example:- lesson scripts greater than 512 characters, would need to be split up before they can be stored as seperate records in the 'dbm' database, and reconstructed when the lesson script is to be retrieved. In the event that the majority of the lesson scripts are greater than 512 characters, the mode of storage would seem better if the lesson files were left as conventional UNIX files rather than as a record in a 'dbm' database.

Problems with porting from a VAX 11/750 to a PDP 11/44

All the programmes were first developed and tested on a UNIX BSD4.1 (VAX 11/750). The C Program Checker, 'lint' was used on the programmes written to detect for features that might cause 'bugs' or other problems that might make the programme difficult to port from one computer to another.
[Johnson, S.C. 1978b, Lint, A C Program Checker]

One feature that 'lint' did not detect was long names that are in danger of being duplicated if truncated to a smaller size word. This occurred in one programme with two long routine names:-

```
     helpmenu1 (...........)
     helpmenu2 (...........)
```

The programme works on a VAX and passes through the 'lint' checks but failed during compilation on a 11/44 with "helpmenu multiply defined" message. Apparently routine names on UNIX VII should be unique in its first 8 characters used otherwise the compiler gives the message that it is confused over which routines (helpmenu1 or helpmenu2) you are calling upon.

<u>The screen editor (sc) modifications</u>

There are 12 programmes that go to make up 'sc' and 1 programme that sets up the help file for the 'sc' help facility. In all about 7,000 lines of C code. The 'sc' on the VAX and PDP are similar so amendments for one will work on the other.

Two basic modifications are made to the 'sc' programmes to satisfy to some extent the requirements of an editor for the CALUNIX WRITER environment. The first change is the execution and inclusion into 'sc' of the help text pertaining to Writer Commands(CALUNIX LEARN author language). Two programmes named 'makehelp.c' and 'help.c' were modified to cater for this change.

Another modification was the top and bottom ruler margins and messages that the screen editor 'sc' adopts upon execution. Depending on whether the user is editing a lesson or a plan, the relevant messages and ruler margins are displayed, (see Figures 3.7 and 3.8).

3.3.2 Testing and Debugging

"Testing is the process of executing a programme with the intent of finding errors". [Myers79] This definition by Myers, that testing should be to uncover errors, is a change from a commonly held viewpoint that a successful test is one that no errors are found. An interesting comparison is given by Myers of how a project manager in a software project, and a doctor diagnosing a patient, treat the definition of success in their respective tests, illustrates the contrasting view. If no errors are found by the project manager during a test, the test is deemed successful, but if an error is discovered it is deemed to be unsuccessful. However if a doctor were to find a case of peptic ulcer during the laboratory test, the test would have been deemed successful, if nothing were discovered the patient with the symptoms will have spent the cost on the laboratory tests without finding the cause and perhaps the cure for his condition.

A top-down approach to testing has been adopted. Each module is tested and the test expands as additional modules are added till the whole programme is covered. Progressively the system would be tested for errors. This form of testing is exhaustive and very time consuming. Eventually most of the obvious errors are debugged. For large programmes it gets more difficult to find residual errors.

Each module testing done during the development stages verify

i) the input and output;
ii) the instructions executed;
iii) error and signal handling.

An example of this is the testing of the main calling module CALUNIX. Exhaustive testing for expected results from a given input would be possible for this programme as the number of possible outcomes:-

i) command line parameter input sequence;
ii) menu prompt input sequence;
iii) interrupts;
iv) invalid parameters;

are not very large.

For large programmes, like the LEARN component, it would be difficult to perform any exhaustive sequence of tests. In the case of LEARN and the screen editor 'sc' programmes, most of the codes remain the same with slight modifications made in certain modules of the programme. Testing would only be performed on the modified codes and how it may effect the expected function of the system.

In LEARN for example tests were made on the additions to the original UNIX LEARN:-

1) scroll back option of the number of lines displayed is more than a screenful;

2) database retrieval of learn scripts stored in the 'dbm' database

3) help option.

Similarly in the screen editor tests on:-

1) the top and bottom ruler and options messages;
2) help text;

were carried out.

After each error has been identified, the debugging process follows and the cycle of testing, debugging and retesting continues, till the desired result is satisfied.

3.5 CONCLUSION

Some useful lessons were learnt from the development and implementation of the CALUNIX system. Using existing facilities can reduce the amount of coding required, thus saving on time and resources, but certain facilities may prove to be inadequate for the application. Modifications to the facilities could remedy the inadequacies to a certain extent but the danger of creating a nonstandard version of the facility might cause maintenance and portability problems. This could, and did happen, for the 'sc' and LEARN programmes on the local UNIX systems, when new versions were implemented, one as a result of the UNIX upgrade from BSD4.1 to BSD4.2 for the VAX 11/750, and the other when a new version for 'sc' was implemented. Modifying existing programmes is a mixed blessing, on one hand there are useful features and codes which can be reusable in other parts of the system, e.g. signal handling routines, but overheads like understanding fully the workings and debugging the modified programmes to be overcome.

Each main component of CALUNIX, that is, WRITER or LEARN or HELP can be independently executed as a command like any other UNIX commands. In the same manner, future programmes components can be integrated into CALUNIX by making provision for the new component as a new menu option in the main CALUNIX programme.

Costs in terms of resources and time are kept down by using readily available programmes. Incorporating the existing computer aided learning programme (LEARN) has maintained the effectiveness of presenting the subject material and the drill and practice approach of LEARN is flexible enough to teach most subjects. Certain subjects like languages, art and music could not be included unless special equipment can be interfaced and software written to operate this special equipment. The portability of CALUNIX has been shown to be successful when the system is ported over from VAX to PDP. Porting CALUNIX to non-UNIX systems with a resident full C language and C library routines might be challenging work to carry out in the future.

The CALUNIX system in its current state can serve as a prototype for future work. There are features which have been planned but not implemented and certain restrictions that need to be rectified. One main component left out was a facility to maintain the system's files, directories, programmes and documentation. UNIX system's 'MAKE' for maintaining programme groups, and the 'nroff' documentation formatting command can serve as the basic tools to build up a maintain facility[Feldma78].

The check option in CALUNIX WRITER could be improved if more comprehensive pattern and syntax checking are implemented perhaps using the 'awk' pattern scanning and processing language instead of a straightforward C programme. [Aho78]

The screen menu interface could be speeded up if a simple line by line output were implemented instead of using cursor control codes to update, protect and refresh the screen image. The implementation protects certain portions of the screen (top window) from being overwritten by say a user who may type faster than the system can cope with however this protection also meant that refresh rate of the screen image is slowed down and also menus could not be displayed on terminals other than on Televideo's 912 and 910 family of visual display units. This could be rectified if the 'termcap' - terminal capability database is set up or some method similar to 'termcap'.

References

Aho78
Aho, A.V., Kernighan, B.W. and Weinberger, P.J., "Awk-A Pattern Scanning and Processing Language". UNIX Programmer's Manual, Bell Labs., Sept 1978.

Arnold80
Arnold, C.R.C.K., "Screen Updating and Cursor Movement Optimization", "A Library Package", University of California, Berkerley, 1980.

Boulay81
Boulay, B.D. and O'Shea, T., "Teaching Novices Programming" In Computing Skills and the User Interface, pp 143-200, Academic Press, 1981.

Bourne82
Bourne, S.R., "The UNIX System", Addison Wesley, 1982.

Elfrin85
Elfring, G., "Choosing a Programming Language", IN BYTE, Vol 10, No 5, pp 235-240, June 1985.

Feldma78
Feldman, S.I., "Make - A program for maintaining Computer Programs", UNIX Programmer's Manual, Bell Labs., Aug 1978.

Hunter82
Hunter, J.A. and Hall, N.A., "A Network Screen Editor Implementation", SOFTWARE - PRACTICE AND EXPERIENCE, Vol 12, pp 843-856, Jan 1982.

Jackso83
Jackson, M.A., "System Development", Prentice-Hall, 1983.

Joy80
Joy, W., and Horton, M., "An Introduction to display editing with vi", University of California, Berkeley, 1980.

Kernig78
Kernighan, B.W. and Ritchie, D.M., "The C Programming Language", Prentice Hall, 1978.

Kernig79
Kernighan, B.W. and Lesk, M. "LEARN - Computer Aided Instruction on UNIX (Second Edition), Bell Labs., Murray Hill, New Jersey 07974, Jan 1979.

Myers79
Myers, G.J., "The Art of Software Testing", John Wiley & Sons, 1979.

Televi82
Televideo Systems, Inc., "TELEVIDEO: Model 910plus terminal Operator's Manual", Televideo systems, Inc., 1170 Morse Avenue, Sunnyvale, California 94086, 1982.

Thomso74
Thomson, K. and Ritchie, D. "The UNIX time-sharing system", IN UNIX Programmers Manual, 1974.

Wallis82
Wallis, P.J., "Portable Programming, Macmillan Press, 1982.

Wirth75
Wirth, N., "An assessment of the programming language Pascal", IEEE Transactions on Software Engineering, Vol SE1, No 2, pp 192-198, June 1975.

CHAPTER 4

TRENDS AND CONCLUSION

4.1 Trends in Computer Technology

The Fifth Generation Project launched in the fall of 1981 by Japan has sparked off similar projects involving key technologies of Intelligent Knowledge-Based System (IKBS), Man-Machine Interface (MMI), Software Engineering, Very Large Scale Integration (VLSI), New Computing Architectures and Communications. [Uehara83] [Moto083] The Alvey Programme in the United Kingdom, ESPRIT (European Strategic Programme of Research and Development in Information Technology) and DARPA (Defence Advanced Research Projects Agency) in the United States are some of the programmes initiated with similar aims to the Fifth Generation Project.

These new developments in computer technology as well as current trends in software development have given education and training a wider scope and potential. This chapter outlines the current trends and further work that could be done in CAL.

4.1.1 Computers

Chip technology has grown dramatically over the past decades. Since 1965 the number of transistors that could be put on a single chip has doubled every fourteen months or so. The prices of microprocessors have also fallen steadily. The falling prices and increased computer power are met by an increasing market for computers.[O'Shea83]

This is reflected in the way microcomputers have evolved from the 8 bit based microprocessors like Apple IIe, Pet 8032, which predominated in the 1970s to the 16 bit based microprocessors on machines like the IBM PC(Intel 8088), Apple Mackintosh (Motorola 68000) that have been prevalent in the market since the early 1980's. By the 1990's, considering the trend of the 8 and 16 bit microcomputers, the 32 bit microcomputer will likely to take to take over the dominant position the 16 bit microcomputer holds today. [Crecin86]

Lateral developments to the serial type computers, in coping with the growing computation needs are:-

(1) parallel type computers, like the ICL Distributed Array Processors (DAP) and the Inmos Tranputers. Parallel type computers take advantage of application programmes which have a high degree of concurrent processing, like in CAD(Computer Aided Engineering), image processing, finite element analysis, matrix manipulation, telephone switching systems and many others.

(2) optical type computers, like the Heriot-Watt computers, use optical switches based on bistable materials. The bistable materials can be switched between two states,

similar to an electronic switch, depending on the intensity of light shone through them. The optical computers have the potential in performing up to 100 million processes in parallel, mainly because unlike flow of electrons, beams of light do not interfere with each other, thus multiple streams of light can be passed through the same switch. Another advantage is that the speed of optical computers could reach over 1000 times more than computers based on conventional chips.

In the past, computer system performance has regularly increased by a factor of ten each decade due mainly to advances in VLSI technology. To support the Fifth Generation Computer target for computers, that require a 1000 times improvement in present day systems, cannot be fulfilled using projected improvements in VLSI techonology alone. The development of parallel systems like transputers, and optical type computers, have been heralded as the means to achieve the quantum leap in processing preformance necessary for the Fifth Generation systems.[Inmos85][Gostic79]

4.1.2 Mass Storage

One of the main needs of computing is the provision of a reliable, fast and compact means of storing huge amounts of information. In the past, magnetic tape was the practical mass storage medium, now the magnetic disc dominates and the prediction is that optical discs will eventually become a more popular means of mass storage.

Magnetic tape did not fully satisfy the computing requirements, mainly because of its slow speed which made it incapable of meeting the random access requirements of computers, even during the mid fifties. The advent of magnetic disc has allowed access times, well below a second. However the main concern of users is in the reliability. Magnetic disc drives have disc heads that move very close to the disc media when reading or writing data to/from the disc. Though in general head crashes seldom occur, one bad experience would be enough to shatter the confidence of the user. Hence users have tended to rely on backups in anticipation of just such an occasion. The quest for speed and reliability has led to a great deal of interest in optical storage devices.

Optical storage was introduced in 1978 as consumer video system based on a standard called LaserVision. The video images are stored as FM signals on the disc. Later this technology was used to produce the optical audio disc on which audio information is encoded digitally. They were known as compact discs (CDs) perhaps because they were miniaturised versions of the LaserVision discs.

The success of CD supported the introduction of an adaptation of the CD, known as CD-ROM for Compact Disc - Read Only Memory in early 1985. Later Write Once Read Many (WORM) type optical discs appeared, allowing the data to be written only once on the optical disc and the written data could then not be erased or rewritten. The data on the WORM disc could be read over again and again.

The advantages of optical disc technology over the magnetic disc technology in handling digital information are:-

(1) Much higher capacity of information can reside on similar size disc. A factor of 10 can be reached in storing on an optical disc compared with a similar size magnetic disc;

(2) Mass replication of optical discs can be done inexpensively, whereas data on a magnetic disc cannot be mass reproduced;

(3) The optical discs are removable unlike the hard disc, this can be used for archival purposes;

(4) Immune to accidental erasure and external magnetic field.

Disadvantages of the optical disc technology are:-

(1) Currently the media can be written once only.

(2) Access times of optical disc drives are slower than high-performance magnetic disc drives. Access times for optical drives range from 100 to 500 milliseconds by contrast high-performance magnetic discs range from 16 to 30 milliseconds. [Fujita84]

Research on eraseable optical discs is still in progress, on different materials. Magneto-optical material like amorphous magnetic gadolinium-iron-cobalt, has been cited as a promising medium for a future eraseable magneto-optical disc. The magneto-optical medium is heated by a laser which reverses the magnetic polarity of a small area and freezes it in that state. Polarised laser light can then be used to read the pattern of magnetised areas which rotates the direction of polarisation of the reflected light, a phenomenon known as the Kerr effect. Erasing the data involves reapplying the laser on the area while an external magnetic field is applied in the original direction of polarisation. Prototypes of eraseable optical discs like the 'jukebox' type from NEC holds two stacks of 100 discs each and two separate drives with a capacity of 120 gigabytes while occupying a 5 cubic feet space. [Herman86]

The large capacity of WORM type and perhaps future erasable optical discs have made possible the storing of full text or graphical data and large files on a relatively cheap and convenient medium.

Multi-media communication is facilitated with the interfacing of computer with the video disc system. Current video disc systems have allowed facilities to be controlled by the computer like:

(1) random access to individual frames of pictures;
(2) freeze frame;
(3) indexing;
(4) teletext overlaying;
(5) slow/fast reverse/forward motion.

LaserVision or videodisc does provide a finesse to the problems of creating graphical and animated sequences for use in computer assisted learning. This is because present graphical environments like SMALLTALK still do not allow easy means of creating good graphics let alone animated sequences. One likely solution to this might be the integrating video and digital information. One such commercial product launched recently is the Multimedia Interactive Control (MIC) 2000 system using an IBM PC, MIC card and software which allows the mixing of signals from video and the PC. Though the MIC system does not let you change the appearance of the video, it allows sound and vision to fade or increase, select sequences of video and superimposition of graphic images onto a moving video or still. Accompanying software have extended MIC to let the user control the video from within an application's package. [Massey86]

4.2 Trends in Software Tools

Software plays a major role in the usability of computer systems. While the cost of computer hardware becomes less over time due to better production methods, the cost of software has risen steadily, and the trend indicates that software cost will remain relatively high. Another general trend in software is its non-portability over different computing machines . This can be seen in the tendency for large firms to develop propriety systems, in order to differentiate rather than integrate with other software. As technology advances, software generally needs to be modified to compensate for this change. Computer users will no doubt benefit if software could migrate gracefully over to successful generations of more powerful computers. The savings in cost and time, and resources of leaving existing practices by the users and software developers alike, can be realised if portability and stability issues can be resolved by software developers.[Ramamo84]

Operating Systems

Operating systems play a key role in the transition of higher level software it supports. A machine independent operating system like UNIX has been the starting point towards a more unified means of matching stable software with machinery. The M.I.T's Project Athena (described in 4.3) in the United States of America have opted for UNIX as the campus wide operating system for its network of computers. Higher level software on the campus are developed on this basis, leading toward better compatibility and coherence between all software tools.

User Interface

A common user interface, would in the same manner provide the same benefits to the computer user that a common operating system provides for the programmers. The use of windows on the screen, to show the different processes running and interacting with the process via the appropriate windows, have been in increasing use since its introduction by Xerox PARC's Learning Research group. Windows are dependent on use of graphics and graphics based tools.

Increasing interest in the use of windows or iconic based interfaces have spawned a new range of software tools like Software-ICS (Software-Icons) library, object-orientated languages and bit-map editors. [Cox86] [Sun86]

The need for iconic tools is necessary as iconic

iconic programmes are difficult and more complex to build. Iconic programmes need not only do all that conventional programmes do, but they must also present their workings, as picture images rather than just text. They must also determine the different input devices like mouse and/or keyboard requirements of the user. One interesting feature of object orientated (iconic) programming is the reliance on reusable codes.

Database Systems

Object based environments and multimedia applications have spurred the development of DBMS which support these environments. One example is MINOS, a multimedia information system which handles unformatted data such as text, voice and images. [Christ86)

It used to be that only large main-frame database systems provided all the useful software tools that assist in administration and maintenance of a database system. Current trends show that these tools are also being adopted by microcomputer based database systems.

Cullinane's IDMS, on ICL machines is a network type DBMS. The software tools that could be used with the system include:-

(1) A data dictionary System (DDS)

(2) An IDMS Query Language (QueryMaster)

(3) Application development tool (Application Master)

The Data Dictionary System or DDS is a powerful tool in that:-

(a) all data contained in the databases are defined here;

(b) the data definitions are documented in this system;

(c) the data defined (schemas and subschemas) can be utilised by external programmes (e.g. COBOL);

(d) it can be used as a design tool for the database.

A query language provides an alternative and an easier means of interrogating the database without relying on a programming language. There are restrictions in a query language in that it could satisfy most of the reporting needs but not much or any of the processing needs.

Application development tools like the ICL's Application Master augment the construction of programmes to interact with conventional files as well as database systems. The MANTIS application development system, is another example of an application development system from Cincom Systems, that interfaces with a TOTAL database as well as conventional files. The two examples of application development aids mentioned, are mainframe-based systems. Revelation's RDESIGN tools is one example of a micro-computer based application development aid. [Cosmos85] [Cincom81]

4.3 Trends in CAL

Europe

A number of EEC programmes on education have directly or indirectly promoted the use of Computers and Computer Assisted Learning. One of the earlier programmes was the NIT's (New Information Technology and the School System in the European Community 1985-1987). COMETT (Community in Education and Training for Technology 1986-1992) is a more recent programme, which focusses on Industry-University partnerships mainly in the areas of training, exchange of ideas, information and expertise, and promoting open learning systems.

The most recent programme is DELTA (Developing European Learning through Technological Advance). DELTA aims to explore the development and techniques for advanced open learning systems. The areas to be covered by DELTA include

- development of a learning system reference for Europe;
- identification of development requirements and their translation into a work programme for concerted action;
- testing and development of measures - to facilitate the introduction of open learning.

So far the last items mentioned have been tentatively identified and the DELTA project outline proposals are for either shared-cost studies or fully financed studies. [Lewis86]

U.K.

The Computers in Teaching Initiative in the U.K.

The Computer in Teaching Initiative (CTI) started as a result from an 18 month investigation on undergraduate training in the Universities. The Computer Board and the University Grants Committee (UGC) became the main parties involved in the initiative and the subsequent funding.

The report highlighted the potential effectiveness of wider workstation availability. However, reaction to the report by users in the universities indicated that software and liveware for teaching developments should also be provided.

University departments of various disciplines submitted their project proposals, finally 106 projects have been funded and over 7 million pounds(U.K.) were committed as at July 1986.

An interesting phenomenon arising from a survey of the equipment used by the projects, is that over 98% makes use of microcomputers rather than minis or main-frames. Perhaps the size of the grant has the effect of easing out the more costly minis and main-frames from the proposals on the equipment submitted. The dominant make is the IBM PC/PC compatible range of microcomputers, which reflect IBM's dominance in the microcomputer market and perhaps also the great amount and variety of software and other support equipment readily available for the IBM PC range of computers.

Most of the projects are in their early stages thus there is little to report as yet on the progress of the initiative. Progress reports on some of the earlier projects suggest the following:-

(1) microcomputers are frequently used as workstations;

(2) networks especially local area networks (LAN) are used for data transfer and communication;

(3) readily available software are used to provide an environment for students to use the workstations;

(4) research on the application of AI techniques in computer aided teaching are planned;

(5) workshop and Conferences have been initiated and

planned, to facilitate the sharing of the work and problems encountered so far by the individual project groups.

Computer manufacturers like IBM and DEC have also funded individual projects in the educational establishment independent from the CTI projects. [CTISS86]

Though dramatic decrease in the cost per hardware function presently seem noticeable only in small scale computing, main-frames do get more powerful for the same cost, supercomputers get a little more affordable for the esoteric few. Main-frames and supercomputers provide the raw power and capacity that is necessary to academic users as expansion in the type of application grows. Higher education establishments still rely on large main-frames and minis as part of their computing service and trends toward distributed computing grows as local, regional and public network systems get more popular. [James86] [Hartley86] [Dallai84]

Educational Computing in U.S.A.

Microcomputers and networks are the keys to the teaching approach the U.S. educational establishments have adopted. A survey of 15 universities and colleges in the

U.S. in 1984 has indicated the trend towards the use of microcomputers either as standalone systems for the students or as part of a distributed network, linked to main-frames.

Project Athena at MIT (Massachussets Institute of Technology) is one of the more ambitious projects costing US$70 million to link the whole university based on a single operating system, the Berkerley UNIX BSD 4.2. IBM and DEC were the main suppliers. In the first phase DEC supplied 63 networked VAX minicomputers with 4 to 6 terminals each and IBM supplied a distributed system of 500 PC XT's with 32 bit co-processors, high resolution bit mapped displays and local-area network interface cards. The PC XT's are organised into several local area networks, each network is supported by an IBM 4341 as a file server and a laser printer. Software is based on UNIX BSD 4.2 which includes C, FORTRAN, LISP and Pascal programming languages, editor and printer formatter. A main theme on the project is coherence in that certain standards are required of the users when developing programmes on the Athena network. Coherence is aimed at preserving the investment in educational software and limiting the training cost. A high level abstraction is promoted in applications development and consistency in the interfaces. [Osgood84] [Balkov85]

Similar activities in providing an integrated campus wide computing environments, have been noted in other U.S.

campuses, like Carnegie-Mellon University and Clarkson University. Other universities like Stanford and Michigan have opted for a heterogeneous environment of different makes of equipment and standards. The argument for a heterogeneous environment is that no single make of equipment could meet the needs of the campus community.

4.4 CONCLUSION

The success of computer aided learning systems in the long term can be measured in the contribution it makes to changing the methods of instruction. A number of barriers have to be overcome:-

(1) resistance to change is a major factor against establishing CAL;

(2) costs of implementing new methods and equipment;

(3) lack of major breakthroughs in new teaching systems;

(4) incompatibility of different CAL courseware.

Resistance to change can be attributed to fear of losing jobs, lack of understanding the new technology and dislike in changing existing practices. Introducing computer literacy courses and computers to all levels of the population have been concerted to bring about a better understanding of computers, their uses and hopefully wider acceptance. Laws have been passed to protect the privacy of individuals and restrict the misuse of the new technology. Creating better understanding and alleviating known fears hopefully will bring about changes in attitudes towards computers.

Economic, social and political pressures are dominant influences which are much more intractable than technological problems. Countries that need to tackle basic problems of poverty, unreliable electrical supply and communications, need not have to worry or seriously consider the implications of new technology.

Artificial intelligence has a history of over 20 years, it's influence in CAL have gained some ground in the area of expert systems approach to teaching. In areas like medicine, geography and mathematics, expert knowledge is more easily identified than other areas like law, psychology and other subject areas that encompass a number of diverse opinions and contradictions. However, optimism still prevails as a number of research projects are still carried out in artificial intelligence methods.

Expert Systems in teaching or Intelligent Tutoring Systems (ITS) has abandoned the early CAI's (Computer Assited Instruction) objectives, that is of providing total courses, and has concentrated on building systems which provide supportive environments for more limited topics. The transfer of knowledge and problem solving skills communicated by human tutors, are done so implicitly. Thus much of the expertise in a particular field has never been articulated. It resides in the hands of the tutors, getting there through experience, abstracted but not necessarily accessible in an articulatable form. Designers of ITS would need to make knowledge explicit as current computer based coaches is limited in learning through experience. The hope now is for educational theories to discover explicit formulation of tutoring, explanation and diagnostics processes inherent in ITS, providing a test bed for developing more precise theories of teaching and learning. Since ITS is an activity based learning, it is unlikely that the computer based tutor will be able to handle all situations. Creating a learning environment to encourage individual members to help one another provide a congenial and effective backup for these systems. A helpful environment helps to break down the competitive or "test taking" mentality. More attention would be focused on student modelling and diagnostics, but progress will be slow because the motivations and plans underlying a person's behaviour when attempting to solve a non trivial problem can be complex. The rewards are therefore high for any progress made in this area with ramifications extending beyond ITS into the area of diagnostic testing. The feeling of achievement in ITS provide some cause for optimism.

Artificial Intelligence (AI) languages like Lisp, Prolog, Planner help the designers of ITS in exploring and developing the knowledge based software. There is still some debate on which AI language is the most suitable. Some argue that AI languages are in general too precise for implementing what is essentially an imprecise or fuzzy method of gathering knowledge. Nevertheless the Prolog language seems to be gaining in popularity. It has been adopted as the AI language for the Fifth Generation Project (Japan) and a number of ITS workers in the U.K. and Europe have also adopted Prolog. The introduction of cheaper and better Artificial Intelligence languages like Turbo Prolog that runs on microcomputers will promote and sustain the interest and work in expert systems. [Borlan86]

Course material for CAL has been implemented from a diverse range of author and programming languages. This has made the large resource of CAL material incompatible with each other and no doubt effort is wasted due to duplication of effort. If an author language could be adopted as a standard and is portable over a variety of machines, the impact on the development of courseware would be greater than the current diverse CAL software.

Computer Aided Learning activity is increasing but the general methods, software and courseware used and developed seem to and would continue to be as divergent and varied as the way application systems are going.

References

Balkov85
Balkovich, E., Lerman, S., Parmelee, R.P., "Computing in Higher Education: The Athena Experience", Comm. of the ACM, Vol 28, No 11, pp 1214-1224, Nov 1985.

Borlan86
Borland Inc., "Turbo Prolog", Borland Inc., 1986

Christ86
S. Christodulakis, F. Ho, M. Theodoridou, " The Multimedia Object Presentation Manager of MINOS: A Symmetric Approach", Proceedings of ACM SIGMOD 86, Vol 15, No 2, June 1986.

Cincom81
Cincom Systems Inc., "MANTIS - Application Development System", Cincom Systems Inc., 1981.

Cosmos85
Cosmos Inc., "REVELATION Users' Guide Release G", Cosmos Inc., 1985.

Cox86
Cox, B. and Hunt, B. "OBJECTS, ICONS AND SOFTWARE-ICS", BYTE, Vol 11, No 8, pp 161-176, Aug 1986.

Crecin86
Crecine, J.P., "The Next Generation of Personal Computers", EDUCOM Bulletin, Vol 21, No 1, pp 2-10, Sprint 1986.

CTISS86
CTISS, "The CTISS FILE", SWURCC, No 1, July 1986.

Dallai84
Dallaire G., "American Universities need greater access to supercomputers", Comm of theACM, Vol 27 No4, pp292-298, Apr 1984.

Fujita84
Fujitani, L., "Laser Optical Disk: The Coming Revolution in on-line storage", Comm of the ACM, Vol 27, No 6, pp 546-554, June 1984.

Gostic79
Gostick, R.W., " Software and algorithms for the Distributed -Array Processors", ICL Technical Journal, Vol 1, No 2, pp116-135, May 1979.

Hartley86
Hartley, D.F., "The University Computing Service in the late 1980s", University Computing, Editor Dr R F Smith, Vol 8, no 1, pp 20-25.

Herman86
Herman, G., "CD-ROMS - The Future of Mass Storage?", Electronics Today, pp 22-25, October 1986.

Inmos85
Inmos , "Technical Overview: Transputer Architecture", Inmos Ltd., Sept. 1985.

Lewis86
Lewis, R., "Editors: New Information Technologies in Britain and Europe", Journal of Computer Assisted Learning, Vol 2, No 2, Blackwell Scientific, July 1986.

Massey83
Massey, R. "Commentary: Personal Computers and Videotex", BYTE, Vol 18, No 7, pp 114-129, July 1983.

Massey86
Massey, J. "Multi-Media Interactive Video Show", PC UK edition Vol 3, No 9, pp 56-60, Sept 1986.

Moto-O83
Moto-Oka, T., "Overview and Introduction to the Fifth Generation" Japan-Singapore Institute of Software Technology Seminar, Nov 1983.

Osgood84
Osgood, D., "A Computer on every desk", BYTE, Vol 9, No 9, pp 162-184, June 1984.

O'Shea83
O'Shea, T. and Self, J. Chapter 2 "The Next Decade", "LEARNING AND TEACHING WITH COMPUTERS", Harvester Press, pp 245-268, 1983.

Ramamo84
Ramamoorthy, C.V., Prakash, A., Tsai, W. and Usuda, Y. "Software Engineering: Problems and Perspectives", IEEE Computer, pp191-209, Oct 1984.

Uehara83
Uehara, T. 'FUTURE PROSPECTS FOR ON-LINE SYSTEMS BASED ON THE INS', Japan Singapore Institute of Software Technology Seminar, Nov 1983.

Sun86
Sun Microsystems Inc, "Windows and Window Based Tools: Beginners Guide", Feb 1986

A P P E N D I X - P R O G R A M M E D O C U M E N T A T I O N

CALUNIX MAIN INCLUDE FILE - CAL44DEF

```
/*  MAIN INCLUDE FILE FOR CALUNIX PROGRAMMES */



#define PDP1144        0       /* 0 - VAX ;  non-zero value for PDP1144 */
#define DEBUG          7
#define WRITERPLAN     1
#define WRITEREDIT     2
#define WRITERCHECK    3
#define WRITERSETUP    4

static char subject[12];

/* CHANGE THE FOLLOWING DESTINATION DIRECTORY PATHS IN YOUR SYSTEM */
char *dir = "/tmp/writer";
char *getlearn = "/usr/local/cok8/learn/learn2";
char *getwriter = "/tmp/writer/writer";
char *helpbase = "/user/staff/nmh/help/helpunix/file";
char lesson[10];
char choice[10];

extern char *dir, *getlearn, *getwriter, *helpbase;
extern char lesson[10];
extern char choice[10];
extern intrpt(), hangup();

/* LOCAL INCLUDE FILE DEFINING THE TELEVIDEO TERMINAL CHARCTERISTICS */
#include "tvitest.h"

/* STANDARD INCLUDE FILE */
#include <stdio.h>
#include <signal.h>
```

```
              CALUNIX  INCLUDE  FILE - TVITEST.H

/*   TELEVIDEO TVI 910 & 912 CURSOR CONTROL CODES */

#define   ESC         ''      /* ESCAPE character preceding each code */

/*---------------HERE   ARE   ALL   STRINGS---------------------*/
#define   SRBLINK     "G6"     /* START Reverse Blink characters  */
#define   ERBLINK     "G7"     /* END Reverse Blink characters  */
#define   SBLINK      "G2"     /* Blink characters     */
#define   EBLINK      "G3"     /* END Blink characters     */
#define   SHIGH       "G4"     /* Reverse Video - Highlight */
#define   EHIGH       "G0"     /* Reverse Video - End Hightlight  */
#define   SUNDERLINE  "G8"     /* START Underline Characters */
#define   EUNDERLINE  "G9"     /* END Underline Characters */
#define   SUNDERLINEB "G:"     /* START Underline & Blink Characters */
#define   EUNDERLINEB "G:"     /* END Underline & Blink Characters */

#define   SPROTECT    ')'      /* Start Protect Field  */
#define   EPROTECT    '('      /* End  Protect Field   */
#define   PROTECTS    '&'      /* Start Screen Protect Mode   */
#define   PROTECTE    '\''     /* End   Screen Protect Mode-single quote  */
#define   SETTAB      '1'      /* Set  TAB to position        */
#define   CLEAR1      ':'      /* Clear Unprotect Fields to Nulls */
#define   CLEAR2      ';'      /* Clear Unprotect Fields to Spaces */
#define   CLEAR3      ','      /* Clear Screen to Half-Intensity Spaces */
#define   CLEAR4      '*'      /* Clear All Data to Nulls */
#define   DISABLE     '#'      /* Disable Keyboard        */
#define   ENABLE      '"'      /* Enable  Keyboard        */
#define   CLRLINE     'R'      /* Clears line and all lines move up 1 line */

/*        TERM 912  */
#define   SHIGH2      'j'     /* Reverse Video - Highlight */
#define   EHIGH2      'k'     /* Reverse Video - End Hightlight  */

/*
         CURSOR   CONTROL   CODES
*/
/*            POSITION  CODES                         */
#define   P1          ' '
#define   P2          '!'
#define   P3          '"'
#define   P4          '#'
#define   P5          '$'
#define   P6          '%'
#define   P7          '&'
#define   P8          '\''          /* single quote need backlash */
#define   P9          '('
#define   P10         ')'
#define   P11         '*'
#define   P12         '+'
#define   P13         ','
#define   P14         '-'
```

```
#define   P15        '.'
#define   P16        '/'
#define   P17        '0'
#define   P18        '1'
#define   P19        '2'
#define   P20        '3'
#define   P21        '4'
#define   P22        '5'
#define   P23        '6'
#define   P24        '7'
#define   P25        '8'
#define   P26        '9'
#define   P27        ':'
#define   P28        ';'
#define   P29        '<'
#define   P30        '='
#define   P31        '>'
#define   P32        '?'
#define   P33        '@'
#define   P34        'A'
#define   P35        'B'
#define   P36        'C'
#define   P37        'D'
#define   P38        'E'
#define   P39        'F'
#define   P40        'G'
#define   P41        'H'
#define   P42        'I'
#define   P43        'J'
#define   P44        'K'
#define   P45        'L'
#define   P46        'M'
#define   P47        'N'
#define   P48        'O'
#define   P49        'P'
#define   P50        'Q'
#define   P51        'R'
#define   P52        'S'
#define   P53        'T'
#define   P54        'U'
#define   P55        'V'
#define   P56        'W'
#define   P57        'X'
#define   P58        'Y'
#define   P59        'Z'
#define   P60        '['
#define   P61        '\\'         /* backlash requires another */
#define   P62        ']'
#define   P63        '^'
#define   P64        '_'          /* underline */
#define   P65        '`'
```

```
#define   P66        'a'
#define   P67        'b'
#define   P68        'c'
#define   P69        'd'
#define   P70        'e'
#define   P71        'f'
#define   P72        'g'
#define   P73        'h'
#define   P74        'i'
#define   P75        'j'
#define   P76        'k'
#define   P77        'l'
#define   P78        'm'
#define   P79        'n'
#define   P80        'o'
#define   P81        'p'
#define   P82        'q'
#define   P83        'r'
#define   P84        's'
#define   P85        't'
#define   P86        'u'
#define   P87        'v'
#define   P88        'w'
#define   P89        'x'
#define   P90        'y'
#define   P91        'z'
#define   P92        '{'
#define   P93        '|'
#define   P94        '}'
#define   P95        '~'
/*
*       CURSOR  ADDRESSING CODES
*/
#define   CURP       '='            /* = part of the ADDRESS CURSOR  */
#define   CURS       '\'           /* ESCAPE part of ADDRESS CURSOR */
#define   CURR       '?'            /* ? part of READ CURSOR POSN */

/*            CURSOR   MOVEMENT               */

#define   HOME                        /* Home Postion (ctrl & up arrow)*/
#define   CURUP      '\013'           /* Move cursor up 1 line */

/*             EDIT MODE               */

#define   EDITS      'k'            /* Start Local Edit Mode */
#define   EDITE      'l'            /* END Edit Mode/Duplex Edit Mode on */
#define   STX        '\002'         /* Start of  Text  */
#define   ETX        '\003'         /* End of text */
#define   SENDTX     'S'            /* Send all text within STX&ETX inc ESC */
```

CALUNIX MAIN PROGRAMME

```
/****************************************************/
/*        THIS IS THE MAIN CALUNIX PROGRAM          */
/*                                                  */
/*          It calls                                */
/*                 (1) Learn  Component;            */
/*                 (2) Writer Component;            */
/*                 (3) Help Component;              */
/*                                                  */
/****************************************************/
/* Contents of cal44def needs to be changed in your system */
#include "cal44def"
main(argc,argv)
char *argv[];
short argc;
{
        char speed[10], lesson[10], subject[12], choice[10], learn[80];
        char writer[80], c;
        short more, n, times;

        /* Initialise all local variables */
        clearbuf(speed,10);
        clearbuf(lesson,10);
        clearbuf(subject,12);
        clearbuf(choice,10);
        clearbuf(learn,80);
        clearbuf(writer,80);

        signal(SIGHUP,hangup);
        signal(SIGINT,intrpt);

        n = times = 0;

        do{
                if (argc==1){
                        if(n<5) calmenu();
                        n = calprompt();
                        fprintf(stderr," n = %d",n);
                        system("sleep 2");
                        if (n>=5) {
                                printf(" %c%sCHOICE UNKNOWN- TRY AGAIN%c%s",ESC,SHIGH,ESC,EHIGH);
                                fflush(stdout);
                                system("sleep 2");
                                clrlower();
                        }
                }

                /* Check for command line parameters */
                if(argc > 5) fprintf(stderr,"Ooo many parameters - calling selection menu");

                if (argc >= 1 && argc <= 4) {
                        if (argc >= 2 && times==0) c=argv[1][0];
                        switch(c)  {

                        case 'L':
                        case 'l':
                                if(argc > 4) strcpy(speed, argv[4]);
                                if(argc > 3) strcpy(lesson, argv[3]);
                                if(argc > 2) strcpy(subject, argv[2]);
                                sprintf(learn,"%s %s %s %s",getlearn, subject, lesson, speed);
                                n = more = system(learn);
                                break;
                        case 'W':
                        case 'w':
                                if(argc > 4) strcpy(lesson, argv[4]);
                                if(argc > 3) strcpy(choice, argv[3]);
                                if(argc > 2) strcpy(subject, argv[2]);
                                sprintf(writer," %s %s %s %s",getwriter, subject, choice, lesson);
                                n = more = system(writer);
                                break;
                        }
                        if (n==1)  {
                                printf("%c%c%c%c",ESC,PROTECTE,ESC,CLEAR4);
                                fflush(stdout);
                                sprintf(learn,"%s %s %s %s",getlearn, subject, lesson, speed);
                                system(learn);
                                more = 1;
                                continue;

                        }
                        else if (n==2)   {
                                sprintf(writer," %s %s %s %s",getwriter, subject, choice, lesson);
                                system(writer);
                        }
                        else if (n==3) {
                                printf("%c%c%c%c",ESC,PROTECTE,ESC,CLEAR4);
                                fflush(stdout);
                                system("./help");
                                system("sleep 1");
                        }
                        else n = 5;
                }
                more = argc =1;
                c = ' ';
                times++;
                n = 0;
                clearbuf(writer,80);
                clearbuf(learn,80);
        }
        while(more) ;

}
```

```
clearbuf(s,n)
char s[];
short n;
{
        short i;
        for(i=0 ;  i <=n ; i++ )   s[i] = ' ';
}

hangup()
{
        exit(1);
}

intrpt()
{
        char response[20], *p;

        signal(SIGINT, hangup);
        printf(" MAIN CALUNIX ROUTINE");
        fflush(stdout);
        write(2, "--Interrupt.0ant to go on?  ", 28);
        p = response;
        *p = 'n';
        while (read(0, p, 1) == 1 && *p != '0)
                p++;
        if (response[0] != 'y')  {
                printf("%c%c%c%c",ESC,PROTECTE,ESC,CLEAR4);
                fflush(stdout);
                exit(1);
        }
        printf("%c%c",ESC,CLEAR2);
        fflush(stdout);
        ungetc('0, stdin);
        signal(SIGINT, intrpt);
        return(1);
}
/*    MAIN   CALUNIX   MENU   */
calmenu()
{
        printf("%c%c%c%c%c%c%c%c",ESC,CLEAR4,ESC,CURP,P1,P1,ESC,SPROTECT);
        printf("                                                                   ");
#ifndef PDP1144
        printf("                     %c%s CALUNIX    MENU %c%s                           "
            ,ESC,SHIGH,ESC,EHIGH);
#endif
#ifdef PDP1144
        printf("                       CALUNIX    MENU                           ");
#endif
        printf("                                                                ");
        printf("      CALUNIX is a Computer Aided Learning environment on the         UNIX System.       ");
        printf("                                                                ");
        printf("      *** The following options are available on CALUNIX :              -                           ");
        printf("                                                                                                    ");
        printf("          (1) LEARN  option - to learn about a subject;                                             ");
        printf("          (2) WRITER option - to write lessons on a subject;                                        ");
        printf("          (3) HELP  option  - to give help on CALUNIX options;                                      ");
        printf("                                                                                                    ");
        printf("                                                                                                    ");
        printf("          Hit  the  BREAK  key  for  option  to  exit  from           CALUNIX.                      ");
        printf("______________________________________________________________________________________________");
        printf("%c%c%c%c",ESC,EPROTECT,ESC,PROTECTS);

}


calprompt()
{
        /* prompt for options required to be filled in */
        extern char lesson[10];
        extern char choice[10];
        short wchoice;

        wchoice = 0;
        printf("  Which Choice(Option)?");
        fflush(stdout);
        clearbuf(choice,10);
        getit(choice);
        if(choice[0] == 'l' ||  choice[0] == 'L' || choice[0] == '1') wchoice = 1 ;
        else if(choice[0] == 'w' ||  choice[0] == 'W' || choice[0] == '2') wchoice = 2 ;
        else if(choice[0] == 'h' ||  choice[0] == 'H' || choice[0] == '3') wchoice = 3 ;
        else if(choice[0] == 'e' ||  choice[0] == 'E' || choice[0] == '4') wchoice = 4 ;
        else wchoice = 0;
        fprintf(stderr,"choice=%c wchoice=%d",choice[0],wchoice);
        system("sleep 1");

        if(wchoice>=1 && wchoice<4) {
                clrlower();
                return(wchoice);
        }
        if(wchoice==0)   wchoice= 5 ; /* wchoice not in range retry */
        if(wchoice==4)   intrpt(); /* wchoice not in range retry */
        return(wchoice);
}

getit(f)
char f[];
{
        short stat, i;
        char c;

        i=0;

        while((c= getchar()) !='0) {
                f[i] = c ;
                i++ ;
        }
        f[i] = ' ';
        stat = 1;
        return(stat);
}

clrlower()
{
        printf("%c%c",ESC,CLEAR2);
        fflush(stdout);
}
```

CALUNIX   WRITER   PROGRAMME

```
/***************************************************************************/
/*                   CALUNIX  -    WRITER MODULE                           */
/*                                                                         */
/*                                                                         */
/*   calls  (1)writer - plan (calls screen editor to edit lesson plan )    */
/*          (2)writer - edit (call screen editor to edit lessons)          */
/*          (3)writer - check(validation of lessons)                       */
/*          (4)writer - setup(install the lessons in directory/dbm for use) */
/*                                                                         */
/*   Arguments in the form - writer subject choice lesson(optional)        */
/*                           max argc=4                                    */
/***************************************************************************/

#include "cal44def"

main(argc,argv)
char *argv[];
{
        extern char  choice[10], lesson[10];
        short wchoice, moretodo, times, status;
        extern char *dir;
        extern hangup(), intrpt();
        char subpath[20];

        moretodo = 1;
        times = 0;


        /*  CHECK IF PERMITTED TO ENTER THE WRITER DIRECTORY */
        if ( checkfile(dir) != 0) {
                fprintf(stderr,"\n ACCESS TO DIRECTORY %s DENIED ",dir);
                exit(1);
        }

        chdir(dir);
        signal(SIGHUP,hangup);
        signal(SIGINT,intrpt);

        if (argc>1) strcpy(subject,argv[1]);
        if (argc>2) times = 1;

        /* 1ST STAGE :- check for command line parameters & writer options  */
        /*             if no option specified display menu and prompt for option */
        while(moretodo) {
                /* if no option specified display menu and prompt for option */

                if(argc == 1 && times==0) {
                        writermenu();
                        wchoice = writerprompt(0);
                        if (wchoice>4) {
                                printf("\nCHOICE UNKNOWN-TRY AGAIN");
                        fflush(stdout);
                        system("sleep 2");
                        times = 0;
                        continue;
                }
                times=1;
        }
        else if(argc == 2 && times==0) {
                writermenu();
                wchoice = writerprompt(1);
                if (wchoice>4) {
                        printf("\nCHOICE UNKNOWN-TRY AGAIN");
                        fflush(stdout);
                        system("sleep 2");
                        times = 0;
                        continue;
                }
                times=1;
        }
        /* Check if subject chosen exists as a directory now */
        if (argc >= 1 && times>=0 && times<2) {
                strcpy(subpath,dir);
                strcat(subpath,"/");
                strcat(subpath,subject);
                if (access(subpath,0) < 0) {
                        printf("\n Subject %s chosen could not be accessed or was",subject);
                        printf(" not created, return to menu.- PLEASE WAIT A MOMENT.\n");
                        fflush(stdout);
                        system("sleep 3");
                        writermenu();
                        wchoice = writerprompt(0);
                        times = 2;
                        moretodo = 1;
                }
                if (wchoice==WRITERPLAN) {   /* if writer plan */
                        moretodo=0;
                        break;
                }
                if (wchoice==WRITEREDIT) {   /* if writer edit */
                        moretodo=0;
                        break;
                }
                if (wchoice==WRITERCHECK) {   /* if writer check */
                        moretodo=0;
                        break;
                }
                if (wchoice==WRITERSETUP) {   /* if writer setup */
                        moretodo=0;
                        break;
                }

```

```
                        if (argc == 3) {
                                if ((strncmp(argv[2],"plan",10)==0) || (strncmp(argv[2],"a",10)==0))  {
                                        wchoice =  1;
                                        moretodo = 0;
                                        break;  /* PLAN OPTION DO NOT NEED LESSONS */
                                }
                                else if ((strncmp(argv[2],"edit",10)==0) || (strncmp(argv[2],"b",10)==0))   wchoice =  2;
                                else if ((strncmp(argv[2],"check",10)==0) || (strncmp(argv[2],"c",10)==0))  wchoice =  3;
                                else if ((strncmp(argv[2],"setup",10)==0) || (strncmp(argv[2],"d",10)==0))  wchoice =  4;
                                else {
                                        fprintf(stdout,"\n  Option %s Not recognised \
                           PLEASE WAIT A MOMENT.\n",argv[2]);
                                        fflush(stdout);
                                        system("sleep 2");
                                        writermenu();
                                        wchoice = writerprompt(2);
                                        continue;
                                }
                                moretodo = 0;
                                times = 3 ; /* added on 15/8/85 so that it will not do this again */
                        }
                }
        }

        /*   2ND  STAGE  :PERFORM WRITER OPTIONS */
        moretodo = 1;
        while(moretodo)    {

                if( subject!=NULL ) {
                        switch(wchoice) {

                        case WRITERPLAN  :
                                moretodo = wplan(subpath);
                                break;

                        case WRITEREDIT  :
                                moretodo = wed(lesson);    /*prompt for which lesson */
                                break;

                        case WRITERCHECK :
                                status = wcheck(lesson);
                                if(status!=0) {
                                        printf("\nWARNING lesson L%s has errors",lesson);
                                        fflush(stdout);
                                        system("sleep 2");
                                }
                                moretodo = 1;
                                break;

                        case WRITERSETUP :
                                status = wsetup();
                                if (status!=0) printf("\n error");
                                fflush(stdout);
                                system("sleep 3");
                                moretodo = 1;
                                break;

                        }
                        if (moretodo!=0)   {
                                writermenu();
                                wchoice = writerprompt(0);
                        }
                }
        }
}

/*      ROUTINE writerprompt    */
writerprompt(seq)
short seq;
{
        /* prompt for options required to be filled in */
        extern char lesson[10];
        extern char choice[10];
        short wchoice;

retry :

        if (seq==0) {
                printf("\n  Which  subject ?");
                fflush(stdout);
                getit(12,subject);
                seq = 1;
        }

        if(subject[0] == '\0' || subject[0] == '\n' || subject[0] == '\ ' ) {
                seq = 0;
                printf("%c%c",ESC,CLEAR4);
                fflush(stdout);
                goto retry;
        }

        if (seq==1) {
                wchoice = 0;
                printf("  Which  Choice(Option) ?");
                fflush(stdout);
                getit(10,choice);
                if((strcmp(choice,"plan"))==0 || (strcmp(choice,"a"))==0) { /* strcmp */
                        wchoice=1;
                        printf("%c%c%c%c",ESC,PROTECTE,ESC,CLEAR4);
                        fflush(stdout);
                        return(wchoice);
                }
```

```
		if((strcmp(choice,"edit"))==0 || (strcmp(choice,"b"))==0) wchoice = WRITEREDIT;
		if((strcmp(choice,"check"))==0 || (strcmp(choice,"c"))==0) wchoice = WRITERCHECK;
		if((strcmp(choice,"setup"))==0 || (strcmp(choice,"d"))==0) wchoice = WRITERSETUP;
		if(wchoice>1&&wchoice<4) {
			seq=2;
			printf("%c%c",ESC,CLEAR2);
			fflush(stdout);
		}
		if(wchoice==0)   wchoice= 5 ; /* wchoice not in range retry */
	}

	if (seq==2) {
		printf("   Lesson Number/name ?");
		fflush(stdout);
		getit(10,lesson);
	}
	system("sleep 1");
	printf("%c%c%c%c",ESC,PROTECTE,ESC,CLEAR4);
	fflush(stdout);
	return(wchoice);
}



getit(n,f)
short n;
char f[];
{
	short stat, i;
	char c;

	i=0;

	if(n==10 || n==12) {
		while((c= getchar()) !='\n') {
			f[i] = c ;
			i++ ;
		}
		f[i] = '\0';
		stat = 1;
	}
	else {
		stat = 0 ;
		fprintf(stdout,"\nERR-getit- value of n = %d(not10or12)",n);  /*ENCOUNTERED EVEN THOUGH n =10*/
		fflush(stdout);
	}
	return(stat);
}

/*********************************************************************/
/*                WRITER   plan   module                           */
/*                                                                    */
/*          The CALUNIX plan option calls the screen editor to        */
/*          (1) display existing lesson plan on a subject;            */
/*       or (2) to edit existing lesson plan;                         */
/*       or (3) to create & edit a new lesson plan;                   */
/*          exit from plan option will return to the WRITER menu      */
/*                                                                    */
/**********************************************************************/
wplan(localdir)    /* subject is STATIC */
char localdir[];
{
        char subjectplan[12], usersub[80], user[80], call[100];
        short status;

        chdir(localdir);
        status = chdir(localdir);

        clearbuf(subjectplan,12);
        clearbuf(usersub,80);
        clearbuf(user,80);
        clearbuf(call,100);

        /* YOU SHOULD BE IN writer's subject DIRECTORY NOW */
        strcpy(subjectplan,subject);
        if( (subjectplan[0]!='\0') && (subjectplan[0]!=' ')) {   /*non emtpy*/

                /* plan file prefixed by plan then followed by subject name*/
                sprintf(usersub,"./plan%s",subjectplan);
                status=checkfile(usersub);  /* 0 if accessible, 1 if not */
        }
        /* CREATE lessonplan file for plan on subject */
        if (status!=0) creat(usersub,0666);


        /* CALLING MODIFIED VERSION OF SC (SCREEN EDITOR) */
        sprintf(call,"wsc %s psc",usersub);


        system(call);
        chdir(dir);
        return(1);
}

checkfile(checkname)
char *checkname;
{
        if ((access(checkname,06))<0) return(1);
        else return(0);
}

writermenu()
```

```
{
        printf("%c%c%c%c%c%c%c%c",ESC,CLEAR4,ESC,CURP,P1,P1,ESC,SPROTECT);
        printf("                                                                                ");

#ifndef PDP1144
        printf("                         %c%s CALUNIX  WRITER %c%s                                 "
            ,ESC,SHIGH,ESC,EHIGH);
#endif
#ifdef PDP1144
        printf("                                CALUNIX  WRITER                                 "
            );
#endif

        printf("                                                                                ");
        printf("      CALUNIX WRITER supports the lesson writer with the foll\
owing set of tools.");
        printf("                                                                                ");
        printf("      (1) First you must specify the subject which you are\
 writing on.          ");
        printf("      (2) Then choose one of the following options :-           \
      ");
        printf("          (a) PLAN  option - to plan lessons on subject;        \
      ");
        printf("          (b) EDIT  option - to create and edit a lesson;       \
      ");
        printf("          (c) CHECK option - to check/validate lesson ;         \
      ");
        printf("          (d) SETUP option - to setup the lesson;               \
      ");
        printf("                                                                \
      ");
        printf("      (3) For option 2(b),(c) & (d) you must give the lesson\
 name/number          ");
        printf("_______HIT_BREAK_KEY_FOR_OPTION_TO_EXIT_________________________\
________________");

        printf("%c%c%c%c%c%c%c%c",ESC,CURP,P14,P76, ESC,EPROTECT,ESC,PROTECTS);
        fflush(stdout);

}

/*******************************************************************/
/*               WRITER   edit   module                            */
/*                                                                 */
/*        edit option calls the screen editor to:-                 */
/*        (1) display existing lesson file on a subject;           */
/*    or  (2) choose to edit existing lesson;                      */
/*    or  (3) choose to create & edit a new lesson file.           */
/*           Exiting  from  edit  returns  to WRITER menu;         */
/*                                                                 */
/*******************************************************************/
```

```
wed(lessonfile) /* subject is STATIC */
char lessonfile[];
{
        char userless[80], call[80];
        short status;
        extern char *dir;

        chdir(dir);
        chdir(subject);
        /* YOU SHOULD BE IN writer's subject DIRECTORY NOW */

        clearbuf(userless,80);
        clearbuf(call,80);

        sprintf(userless,"L%s",lessonfile);
        status=checkfile(userless);

        /* CREATE lesson file on subject */
        if (status!=0) {
                printf("\nCREATING lessonfile status return=%d",status);
                fflush(stdout);
                creat(userless,0666);
                system("sleep 2");
        }


        sprintf(call,"wsc %s wsc",userless);

        /* calling modified screen editor function */

        system(call);
        return(1);
}

/****************************************************************************/
/*                                                                          */
/*                    WRITER  check  module                                 */
/*                                                                          */
/*        Check / Validates the lesson writer commands                      */
/*        are correct in syntax;                                            */
/*        Allow for future additions of checks into programs;               */
/*                                                                          */
/* Valid Writer {  #print, #user, #create filename, #copyin - #uncopyin,    */
/* Commands Used{  #copyout - uncopyout, #pipe - #unpipe, #cmp file1 file2, */
/*              {  #match word, #bad word, #succeed, #fail,                 */
/*              {  #log file or #log, #next.                                */
/****************************************************************************/
wcheck(checkfile)
char checkfile[];

{
        FILE *fp, *fopen();
        extern char *dir;
        char lessonfile[12];

        chdir(dir);
        chdir(subject);

        clearbuf(lessonfile,12);


        sprintf(lessonfile,"L%s",checkfile);

        if((fp=fopen(lessonfile, "r")) == NULL) {
                fprintf(stderr, "%s file can't open ",lessonfile);
                system("sleep 2");
                return(1);                /*prompt for next action*/
        }
        else {
                /* start sequence of Check routines  */

                chkcommand(fp);             /* check for #positions & commands */
                fclose(fp);
        }
        return(0);
}



chkcommand(f)
FILE *f;
{
        char s[100], c;
        short i, linecnt, errorind;

        linecnt = errorind = 0;
        while((fgets(s,100,f)) !=NULL){
                linecnt++;
                printf("\nlinecount:-%d",linecnt);
                fflush(stdout);
                if(s[0]=='#') errorind = hashchk(s,linecnt);
                else {
                        /* check if any # put in position other than in column 0 */
                        for(i=1; i<=100; i++) {
                                if(s[i]=='\n') break;
                                if(s[i]=='#') {
                                        fprintf(stderr,"Warning -# sign found in line %d column %d",linecnt,i);
                                        errorind++ ;
                                }
                        }
                }
        }
        if (errorind > 0) {
                fprintf(stderr," Lesson may contain errors, PLEASE CHECK\n");
                system("sleep 3");
        }
        return;

}

/*   Check if commands are valid           */
hashchk(hashstr,lineno)
char hashstr[];
short lineno;

{
        short match, i;
        char localstr[], cmpstr[];

        printf("\nPERFORMING hashchk");
        fflush(stdout);

        for(i=0; (i<=10 || hashstr[i]==' ' || hashstr[i]=='\0'); i++){

                localstr[i] = hashstr[i];
        }
        localstr[i] = '\0';

        strcpy(cmpstr,localstr);        /*used inconsistently ? */
        /*      (void) strcpy(cmpstr,localstr);    old statement (remove void) */
        match = 1;    /* set match to notmatch */

        while(match) {

                match = strcmp(cmpstr,"#print");
                if (match = 0) break;
                match = strcmp(cmpstr,"#user");
                if (match = 0) break;
                match = strcmp(cmpstr,"#create");
                if (match = 0) break;
                match = strcmp(cmpstr,"#copyin");
                if (match = 0) break;
                match = strcmp(cmpstr,"#uncopyin");
                if (match = 0) break;
                match = strcmp(cmpstr,"#copyout");
                if (match = 0) break;
                match = strcmp(cmpstr,"#uncopyout");
                if (match = 0) break;
                match = strcmp(cmpstr,"#pipe");
                if (match = 0) break;
                match = strcmp(cmpstr,"#unpipe");
                if (match = 0) break;
                match = strcmp(cmpstr,"#cmp");
                if (match = 0) break;
                match = strcmp(cmpstr,"#match");
                if (match = 0) break;
                match = strcmp(cmpstr,"#bad");
                if (match = 0) break;
                match = strcmp(cmpstr,"#succeed");
                if (match = 0) break;
                match = strcmp(cmpstr,"#fail");
                if (match = 0) break;
                match = strcmp(cmpstr,"#log");
                if (match = 0) break;
                match = strcmp(cmpstr,"#next");
                if (match = 0) break;
                /* if still no match after pass  then give error message */

                if (match!=0){
                        fprintf(stderr,"error - %s at line %d - command unknown",cmpstr,lineno);
                        system("sleep 2");
                        break;
                }
        }
        return(match);
}
/*****************************************************************************/
/*                      WRITER   setup  module                               */
/*                                                                           */
/*      prompts (1) if lessons required to be setup on a database(dbm);      */
/*              (2) checks if files > 1blk size (approx 500chars)            */
/*                  performs split if it is greater;                         */
/*              (3) setup files into a database named after subject;         */
/*                                                                           */
/*****************************************************************************/
#define   OK   0
wsetup()
{
        extern char *dir;
        extern hangup(), intrpt();

        FILE *fp, *fopen();
        char parafile[6], tmpstr1[80], tmpstr2[80], call[100];
        char basedir[20], basepag[20];
        short more, status, errorbit;
        int f1 , f2;

        more = 1;
        chdir(dir);
        chdir(subject);


        clearbuf(parafile,6);
        clearbuf(tmpstr1,80);
```

```
	clearbuf(tmpstr2,80);
	clearbuf(call,100);
	clearbuf(basedir,20);
	clearbuf(basepag,20);

	signal(SIGHUP, hangup);
	signal(SIGINT, intrpt);
	while (more) {
		status  = setupprompt();
		if ( status==OK )  {
			/* SETUP  STAGE 1--splitting */
			system("ls L* > xaa");
			strcpy(parafile,"xaa");
			if((fp = fopen(parafile,"a+")) == NULL) {
				printf("\nERROR parameter file %s cannot open",parafile);
				fflush(stdout);
				more = 0;
				errorbit=1;
				continue;
			}
			else {
				putc('\n',fp);
				close(fp);
			}
			strcpy(tmpstr1,dir);
			strcat(tmpstr1,"/split2");
			sprintf(call,"%s xaa",tmpstr1);
			system(call);   /* perform spilt it */
			fflush(stdout);

			/* SETUP STAGE  2 -- load data in database */

			strcat(basedir,subject);
			strcat(basedir,".dir");
			strcat(basepag,subject);
			strcat(basepag,".pag");

			if((f1 = creat(basedir, 0666)) == -1) {
				printf("\nERROR DBASE file %s cannot open",basedir);
				fflush(stdout);
				more = 0;
				errorbit=1;
				continue;
			}

			if((f2 = creat(basepag, 0666)) == -1) {
				printf("\nERROR DBASE file %s cannot open",basepag);
				fflush(stdout);
				more = 0;
				errorbit=1;
				continue;
```

```
                        }

                        system("ls  L* > sfile");
                        strcpy(parafile,"sfile");
                        if((fp = fopen(parafile,"a+")) == NULL) {
                                printf("\nERROR parameter file %s cannot open",parafile);
                                fflush(stdout);
                                more = 0;
                                errorbit=1;
                                continue;
                        }
                        else {
                                putc('\n',fp);
                                close(fp);
                        }
                        strcat(tmpstr2,"pwd ; ../dbase");
                        sprintf(call,"%s %s  sfile ",tmpstr2,subject);
                        errorbit = system(call);   /* perform load it */
                }
                return(errorbit);
        }

}

hangup()
{
        return(0);
}

intrpt()
{
        char response[20], *p;

        signal(SIGINT, hangup);
        write(2, "\nInterrupt.\nWant to go on?  ", 28);
        p = response;
        *p = 'n';
        while (read(0, p, 1) == 1 && *p != '\n')
                p++;
        if (response[0] != 'y') {
                printf("%c%c%c%c",ESC,PROTECTE,ESC,CLEAR4);
                fflush(stdout);
                printf("\n EXITING  FROM   CALWRITER");
                fflush(stdout);
                system("sleep 2");
                exit(1);
        }
        ungetc('\n', stdin);
        signal(SIGINT, intrpt);
        printf("%c%c",ESC,CLEAR2);
        fflush(stdout);
```

```
}

setupprompt()
{
        char response[20], *p;

        printf("%c%c",ESC,CLEAR2);
        fflush(stdout);

        printf("\n THIS  OPTION setup SHOULD ONLY BE USED WHEN YOU HAVE :-");
        printf("\n\n     (1) created the necessary lessons;");
        printf("\n\n     (2) you wish to set it up on a database(dbm);");
        printf("\n\nTHE LESSONS WOULD WORK JUST AS WELL IF IT WERE LEFT AS IT IS");
        fflush(stdout);

        write(2, "\nWant to go on?  ", 18);
        p = response;
        *p = 'n';
        while (read(0, p, 1) == 1 && *p != '\n')
                p++;
        if (response[0] != 'y') {
                printf("%c%c%c%c",ESC,PROTECTE,ESC,CLEAR4);
                fflush(stdout);
                return(1);
        }
        ungetc('\n', stdin);
        printf("%c%c",ESC,CLEAR2);
        fflush(stdout);

        return(OK);
}

clearbuf(fname,t)
char fname[];
short t;
{
        short i;

        for(i=0; i<=t; i++) fname[i] = '\0';

}
```

```
                        case 'w':
                        case '2':
                                more = writertxt();
                                break;
                                /*   UNIX COMMAND LISTINGS   */
                        case 'U':
                        case 'u':
                        case '3':
                                more = helpunix();
                                break;
                        }
                }
                /*  NO PARAMETERS PASSED */
                if (n==1)  {
                        more = learntxt();
                        continue;

                }
                else if (n==2) {
                        more = writertxt();
                }
                else if (n==3) {
                        more = helpunix();
                }
                else n = 5;
                clrlower();
                more = 1;
        }
        while(more) ;

}

clearbuf(s,n)
char s[];
short n;
{
        short i;
        for(i=0 ;  i <=n ; i++ )   s[i] = '\0';
}

hangup()
{
        exit(1);
}

intrpt()
{
        char response[20], *p;

        signal(SIGINT, hangup);
        printf("\n\n MAIN  HELP  ROUTINE");
```

```
	fflush(stdout);
	write(2, "--Interrupt.\nWant to go on?  ", 28);
	p = response;
	*p = 'n';
	while (read(0, p, 1) == 1 && *p != '\n')
		p++;
	if (response[0] != 'y')  {
		wrapup();
	}
	ungetc('\n', stdin);
	signal(SIGINT, intrpt);
	printf("\n  Which Choice(Option)?");
	fflush(stdout);
	return(1);
}

clearscreen()
{
	printf("%c%c%c%c",ESC,PROTECTE,ESC,CLEAR4);
	fflush(stdout);

}

clrlower()
{
	printf("%c%c",ESC,CLEAR2);
	fflush(stdout);
}

/*	ROUTINE helpprompt	*/
helpprompt()
{
	/* prompt for options required to be filled in */
	extern char choice[10];
	short wchoice;

	wchoice = 0;
	printf("\n  Which Choice(Option)?");
	fflush(stdout);
	getit(10,choice);
	if((choice[0]=='l') || (choice[0]=='L') || (choice[0]=='1')) wchoice = 1 ;
	else if((choice[0]=='w') || (choice[0]=='W') || (choice[0]=='2')) wchoice = 2 ;
	else if((choice[0]=='u') || (choice[0]=='U') || (choice[0]=='3')) wchoice = 3 ;
	else if((choice[0]=='e') || (choice[0]=='E') || (choice[0]=='4')) wrapup();
	system("sleep 1");

	if(wchoice>=1 && wchoice<4) {
		clrlower();
		return(wchoice);
	}
	if(wchoice==0)	wchoice= 5 ; /* wchoice not in range retry */
```

```
        if(wchoice==4)   intrpt(); /* wchoice not in range retry */
        system("sleep 1");
        return(wchoice);
}

getit(n,f)
short n;
char f[];
{
        short stat, i;
        char c;

        i=0;

        if(n==10 || n==12) {
                while((c= getchar()) !='\n') {
                        f[i] = c ;
                        i++ ;
                }
                f[i] = '\0';
                stat = 1;
        }
        else {
                stat = 0 ;
                fprintf(stdout,"\nERR-getit- value of n = %d(not10or12)",n);  /*ENCOUNTERED EVEN THOUGH n =10*/
                fflush(stdout);
        }
        return(stat);
}

hmenu()
{
        printf("%c%c%c%c%c%c%c%c",ESC,CLEAR4,ESC,CURP,P1,P1,ESC,SPROTECT);
        printf("                                                                        ");
#ifndef PDP1144
        printf("                    %c%s CALUNIX   HELP  %c%s                            "
            ,ESC,SBLINK,ESC,EBLINK);
#endif
#ifdef PDP1144
        printf("                       CALUNIX   HELP                                 ");
#endif
        printf("                                                                        ");
        printf("      CALUNIX  HELP  provides the following options          \
                ");
        printf("                                                                        ");
        printf("                                                          \
                   ");
        printf("                                                                \
            ");
        printf("          (1) LEARN  option - Help about learn;                 \
            ");
        printf("          (2) WRITER option - Help on Writer;                  \
                ");
        printf("          (3) UNIX  option  - Help on UNIX commands.           \
                ");
        printf("                                                               \
                ");
        printf("                                                               \
                ");
        printf("          Hit  the  BREAK  key  for  option  to  exit  from   \
CALUNIX.            ");
        printf("_______________________________________________________________\
________________");
        printf("%c%c%c%c",ESC,EPROTECT,ESC,PROTECTS);

}

learntxt()
{
        int stat=0;
        clearscreen();
        fprintf(stdout,"\n Please wait - while learn text is being fetched.\n");
        fflush(stdout);
        if(fork()==0) {
                execl("/bin/sh","sh","-c","man learn",0);
        }
        wait(&stat);
        if(stat<0 || stat >0) fprintf(stderr,"\n ERROR in execl (man learn).");
        system("sleep 2");
        return(stat);
}

writertxt()
{
        int i;
        clearscreen();
        fprintf(stdout,"\n Please wait - while writer text is being fetched.\n");
        fflush(stdout);
        i = system("man writer");
        system("sleep 3");
        return(i);
}

helpunix()
{
        int i;
        clearscreen();
        fprintf(stdout,"\n Please wait for  help on UNIX program to run.\n");
        fflush(stdout);
        i = system("./l");
        system("sleep 3");
        return(i);
```

```
}

wrapup()
{
        clearscreen();
        printf("\n              EXITING FROM  CALUNIX  HELP -- bye \n\n");
        fflush(stdout);
        exit(0);
}

```

CALUNIX   HELP-LIST PROGRAMME

```
/**************************************************************************/
/*                                                                        */
/*                      CALUNIX  HELP list on UNIX                        */
/*                                                                        */
/**************************************************************************/
/*
* List either all of the whatis unix commands or only those selected
* from the database file.
*/
#include "cal44def"
#define pdp1144  1
/*  #define vaxberk  1  */
main(argc, argv)
char *argv[];
{
        char  helpstr[80];
        int   times, ltype;
        extern char  *helpbase;
        short i,j,j10,j20,j30,j40,j50,j60,j70,j80,respond;
        char  *p,*q;
        typedef struct {
                char *dptr;
                int  dsize;
        }
        datum;
        datum key, content;
        datum fetch();
        datum firstkey();
        datum nextkey();

        j =j10 = j20 = j30 = j40 = j50 =  j60 = j70 = j80 = 0;

        /* printf("\n argc=%d & argv[1]=%s",argc,argv[1]);  */
        if (strcmp(argv[1],"1")==0)  ltype = 1;
        if (strcmp(argv[1],"2")==0)  ltype = 2;
        if (strcmp(argv[1],"3")==0)  ltype = 3;
        else if (argc<=1) ltype = 0;

        clearbuf(helpstr,80);

        if(times == 0 ) {
                dbminit(helpbase);

                times++;

                clearscreen();

#ifdef  pdp1144
                if (argc > 2 && ltype== 1 ) {
                        key.dsize = strlen(argv[2]);    /* SELECTIVE LISTS OF KEYED COMMAND */
                        key.dptr  = argv[2];
                        content   = fetch(key);
                        p = content.dptr;
                        i = 0;
                        for (i=0; i < content.dsize ; i++) {
                                putc(*p++,stdout);
                                fflush(stdout);
                        }
                        system("sleep 2");
                        return(i);
                }
#endif
#ifdef vaxberk
                if(ltype ==  1 ) {
                        sprintf(helpstr,"whatis %s",argv[2]);   /* FOR BSD UNIX whatis COMMAND USED */
                        /* printf("helpstr=%s",helpstr); */
                        system(helpstr);
                        fflush(stdout);
                        system("sleep 3");
                }
#endif

                if(argc ==1  && ltype== 0 )   {
                        for (key = firstkey(); key.dptr != NULL; key = nextkey(key)){
                                j++;
                                if (j <= 20) {
                                        move(j,1);
                                        fflush(stdout);
                                }
                                else if ( j >= 20 && j < 40) {
                                        j10++;
                                        move(j10,10);
                                        fflush(stdout);
                                }
                                else if (j >= 40 && j < 60) {
                                        j20++;
                                        move(j20,20);
                                }
                                else if (j >= 60 && j < 80) {
                                        j30++;
                                        move(j30,30);
                                }
                                else if (j >= 80 && j < 100) {
                                        j40++;
                                        move(j40,40);
                                }
                                else if (j >= 100 && j < 120) {
                                        j50++;
                                        move(j50,50);
                                }
                                else if (j >= 120 && j < 140) {
                                        j60++;
                                move(j60,60);
                        }
                        else if (j >= 140 && j < 160) {
                                j70++;
                                move(j70,70);
                        }
                        else if ( j > 160) {
                                move(21,1);
                                printf("Type y to continue/type command if description required");
                                fflush(stdout);
                                respond = gets(helpstr);
                                if (helpstr!='n' && helpstr[0] != 'y') goto  select;  /* good place to try  fork */
                                if(helpstr[0] == 'y' )  {
                                        j = j10 = j20 = j30 = j40 = j50 = j60 = j70 = 0;
                                        clearscreen();
                                }
                        }
                        q = key.dptr;          /* THIS LISTS ONLY THE COMMAND KEYWORDS IN /BIN*/
                        for(i=0; i < key.dsize; i++) {
                                putc(*q++,stdout);
                                fflush(stdout);
                        }
                }
        }


        if (ltype ==  3 ) {
                for (key = firstkey(); key.dptr != NULL; key = nextkey(key)){
                        content = fetch(key);   /* THIS LISTS ALL THE COMMAND CONTENTS */
                        p = content.dptr;
                        for(i = 0; i < content.dsize; i++)  {
                                putc(*p++,stdout);
                                fflush(stdout);
                        }
                        if (j > 14) {
                                move(21,1);
                                printf("  Will continue in 2 secs if not interrupted");
                                fflush(stdout);
                                system("sleep 2");
                                clearscreen();
                                j = 0;
                        }
                        j++;
                }
                system("sleep 2");
        }
        system("sleep 2");
        return(i);

select :
        key.dsize = strlen(helpstr);  /* SELECTIVE LISTS OF KEYED COMMAND */
```

```
                        key.dptr  = helpstr;
                        content   = fetch(key);
                        p = content.dptr;
                        i = 0;
                        move(22,2);
                        fflush(stdout);
                        for (i=0; i < content.dsize ; i++) {
                                putc(*p++,stdout);
                                fflush(stdout);
                        }
                        system("sleep 2");
                        return(i);
                }
        }

        move(row,col)
        short row, col;
        {
                fprintf(stdout,"%c%c%c%c",CURS,CURP,31+row,31+col);
                fflush(stdout);
        }

        clearscreen()
        {
                printf("%c%c",ESC, CLEAR4);
                fflush(stdout);
        }

        clearbuf(s,n)
        char s[];
        short n;
        {
                short i;
                for(i=0 ;  i <=n ; i++ )   s[i] = '\0';
        }

        errorrow()
        {
                fprintf(stdout,"\n Row specified out of the screen's range.");
                exit(1);
        }

        errorcol()
        {
                fprintf(stdout,"\n Column specified out of the screen's range.");
                exit(1);
        }
```

CALUNIX HELP PROGRAMME

```
/***************************************************************/
/*         THIS IS THE CALUNIX  HELP  PROGRAM    v1.2 6/86    */
/*                                                             */
/*         options available are                               */
/*                (1) LEARN  ;                                 */
/*                (2) WRITER ;                                 */
/*                (3) UNIX.                                    */
/*                                                             */
/***************************************************************/
#include "cal44def"
main(argc,argv)
char *argv[];
short argc;
{
        char choice[10];
        char writer[80],learn[80], c;
        short more, n;

        /* Initialised variables */
        clearbuf(choice,10);
        clearbuf(writer,80);
        n = 0;

        signal(SIGHUP,hangup);
        signal(SIGINT,intrpt);

        do{
                if(argc > 5) fprintf(stderr,"\nToo many parameters - calling selection menu");
                if (argc==1 || argc>5){
                        if(n<5) hmenu();
                        n = helpprompt();
                        if (n>=5) {
                                printf("\n   CHOICE UNKNOWN- TRY AGAIN");
                                fflush(stdout);
                                system("sleep 2");
                        }
                }

                /* ARGUMENTS PASSED TO HELP */
                else if (argc > 1 && argc <= 4) {
                        c=argv[1][0];
                        fprintf(stderr," value of c(1st letter of argv1 is- %c\n",c);
                        system("sleep 3");
                        switch(c)  {
                                /*   LEARN HELP TEXT */
                        case 'L':
                        case 'l':
                        case '1':
                                more = learntxt();
                                /*   WRITER HELP TEXT */
                        case 'W':
```

CALUNIX HELP SETUP makefile

```
#   makefile    for CALUNIX help database setup
#

all:    compile1  compile2 compile3 setup  load  clean

compile1:       dbase
        cc -o dbase dbase.c -ldbm

compile2:       setup1
        cc -o setup1 setup1.c

compile3:
        cc -o tidy tidy.c

setup:
        setup1 < cmdfile

load:
        dbase file < cmdfile

clean:
        -tidy < cmdfile
```

CALUNIX   HELP   dbase PROGRAMME

```
/***************************************************************************/
/*      CALUNIX dbase.c  - Loads contents of a file as a record in the dbm  */
/*                         database.                                        */
/*      "dbase  database <  filename(containing record entry filenames "    */
/*                                                                          */
/*                                                                          */
/***************************************************************************/
#include <stdio.h>
char line[4096];
main(argc, argv )
int argc;
char *argv[];
{

        char keyline[20];
        int   len;
        FILE *fp, *fopen();
        typedef struct {
                char *dptr;
                int  dsize;
        }
        datum;
        datum key, content;

        dbminit(*++argv);


        while ((key.dsize = getline(keyline))!= 0) { /* read the key */
                key.dptr  = keyline;


                if((fp = fopen(keyline, "r")) == NULL){
                        printf("can't open %s0, keyline);
                }
                else {
                        content.dsize = readin(line, fp); /* read file into 'line' */
                        content.dptr  = line;
                        store(key, content);
                        fclose(fp);
                }
        }
}
readin(buf, fptr)
char buf[];
FILE *fptr;
{
        int c, i ;
        char  *p;
        p = line;
        i = 0;
        while((c = getc(fptr)) != EOF){
                *p++ = c;
                i++;
        }
        *p = ' ';
        return(i);
}


getline(s)
char s[];
{
        int c, i;
        for (i=0; (c=getchar())!='0; ++i) s[i] = c;
        s[i] = ' ';
        return(i) ;
}

```

CALUNIX HELP setup1 PROGRAMME

```
/***********************************************************************/
/*        CALUNIX  setup1.c  -  One time setup programme for help dbm    */
/*                              database. (uses BSD4.1 "man -k" or       */
/*                              "whatis" commands to generate files      */
/*                               containing the help text).              */
/*                                                                       */
/*        " setup1 < filename(containing unix commands)  "               */
/*                                                                       */
/***********************************************************************/

#include <stdio.h>
char line[1024];
char keyline[20];
main()
{
        char command[80];
        extern char keyline[20];
        int   len;
        FILE *fp, *fopen();


        while ((len = getline(keyline))!= 0) { /* read the key */
                sprintf(command,"whatis %s > %s",keyline,keyline);
                system(command);
        }
}

getline(s)
char s[];
{
        extern char keyline[20];
        int c, i;
        for (i=0; (c=getchar())!='0; ++i) s[i] = c;
        s[i] = ' ';
        strcpy(keyline,s);
        return(i) ;
}

```

CALUNIX HELP tidy PROGRAMME

```
/***************************************************************************/
/*       CALUNIX tidy.c  - removes unwanted files left from setup1         */
/*                                                                         */
/*        " tidy < filename(file containing the unix help commands) "      */
/*                                                                         */
/***************************************************************************/
#include <stdio.h>
char line[1024];
char keyline[20];
main()
{
        char command[80];
        extern char keyline[20];
        int   len;
        FILE *fp, *fopen();


        while ((len = getline(keyline))!= 0) { /* read the key */
                sprintf(command,"rm %s ",keyline);
                system(command);
        }
}

getline(s)
char s[];
{
        extern char keyline[20];
        int c, i;
        for (i=0; (c=getchar())!='0; ++i) s[i] = c;
        s[i] = ' ';
        strcpy(keyline,s);
        return(i) ;
}

```