AN ELECTROCARDIOGRAPH TUTOR
USING
EXPERT SYSTEM TECHNOLOGY

Derek Louis Meyer

A thesis submitted to the Dept. of Biomedical Engineering, Faculty of Medicine, University of Cape Town, in partial fulfilment of the requirements for the degree of Master of Science (Biomedical Sciences).

April 1988

ERRATA

Page	vii	line 22	:	spacing error
"	viii	line 14	:	spacing error
"	1	line 19	:	"an insight"
"	17	line 19	:	"Spiegelhalter"
"	25	line 14	:	"Spiegelhalter"
"	32	line 23	:	"many of the computers"
"	33	line 6	:	"The the 63000 system"
"	33	line 19	:	"an in-house system"
"	36	line 13	:	"it is unlikely"
"	50	fig 5.1	:	"Analog"
"	54	line 2	:	"signal."
"	54	line 3	:	"it must also"
"	61	line 16	:	"which the make it"
"	61	line 22	:	"student can is also urged"
"	87	fig 6.7	:	"base you are interested in"
"	91	line 3	:	"abnormality"

ABSTRACT

Computer systems for the interpretation of diagnostic ECGs are widely used, but currently provide no explanatory or teaching functions of value to the less experienced practitioner. The relevant literature is reviewed, and specifications are provided for an ECG analysis system which will function as a learning aid for undergraduate and postgraduate medical students. Key aspects of the specifications are implemented on an IBM-PC. Recommendations for further development are provided.

DECLARATION

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science at the University of Cape Town. It has not been submitted before for any degree or examination at any other University.

Signed by candidate

Derek Louis Meyer.

27th day of April, 1988.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

A-D	Analog to digital
AHA	American Heart Association
AVA	American Veteran's Administration
ECG	Electrocardiograph
kb	kilobyte
MSDOS	Microsoft Disk Operating System
RAM	Random access memory
UCT	University of Cape Town
UCSD	University of California, San Diego

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Existing expert systems for the computer analysis of ECGs can provide the physician with a high quality ECG tracing complete with interpretation in a few minutes. Such systems, however, have the disadvantage of providing very little explanation or teaching of value to the less experienced practitioner.

A system which could compete with existing computerised ECG analysis systems in terms of quality, and in addition provide tutoring facilities for students, both undergraduate and postgraduate, who wish to test and improve their knowledge of ECGs in a way that cannot be done well from a book would overcome this drawback. It should have a place in continuing medical education among non-cardiologist practitioners at all stages in their careers.

This thesis is concerned with establishing the feasibility of such a system, and presents recommendations on how development of the system should proceed.

Alternate approaches in expert systems technology are critically discussed. A review of important projects that apply expert system technology to clinical medicine provides a insight into the strengths and shortcomings of the technology. However, computer science is a fast moving discipline. New developments may render existing technology obsolete, and force us to change our opinions. One such development, the construction of simulated neuron networks, may revolutionise the field of expert systems, and is briefly discussed.

Systems to analyse diagnostic ECGs are reviewed, and a specification for the proposed ECG Tutor is provided. Key aspects of the specification are implemented in a prototype system, running on an IBM-PC. The prototype system demonstrates the feasibility of the specifications.

Conclusions drawn from this work are presented, and give a clear indication of how best to proceed with the development of the ECG Tutor.

CHAPTER 2

AN INTRODUCTION TO EXPERT SYSTEM TECHNOLOGY.

2.1 A SHORT HISTORY OF ARTIFICIAL INTELLIGENCE RESEARCH.

The concept of 'intelligence' has long presented philosophical difficulties. The original meaning of the word was simply 'knowledge', as in 'military intelligence', the higher functions being refered to as 'spirit' or 'soul'. Our present understanding of the term as a measurable quantity, implicit in IQ testing, developed in the seventeenth century.

The concept of "machines that think" has been entrenched in popular literature, and computer scientists are investigating the possibility of creating fact from science fiction.

Early research on Artificial Intelligence (AI) concentrated on finding general solutions to mimic intellectual activity, which could then be applied to any problem domain. Newell developed a "General Problem Solver - A Program that Simulates Human Thought" and Adelson developed an "Ideology Machine" (Boden 1977). These systems did not live up to their titles, and work in this direction failed to excite more practically minded scientists.

By the early 1970s attempts at general solutions were largely abandoned in favour of systems that could operate competently in limited domains. This change of focus had one distinct advantage. The vexing questions of "How do we think?" and "What is intelligence?" could be safely ignored. Much attention was focused on well defined problems such as games.

2.2 EXPERT SYSTEMS.

A system that can operate competently in a limited domain can be considered to be an "expert" in that domain, i.e. it is performing a function which would require expertise if performed by a person. Expertise is essentially specialist knowledge. Although digital computers have proved useful tools for the manipulation of data, there is a distinction between data and knowledge. Attempts at producing systems which perform the functions of human experts highlight this distinction. Work on expert systems has tended to concentrate on methods for representing and manipulating knowledge in a computer. Intelligent machines are machines that know - an unexpected return to an earlier concept of intelligence.

Expert system researchers have worked in a wide range of disciplines. For example, systems have been produced which give advice on medical diagnosis, geological exploration and computer configuration. One writer claims that "....expert systems are changing the personal computer's role from trusty assistant to wise consultant." (Lemley 1985).

Expert systems are arousing much interest in industry and elsewhere. It is envisioned that they will be able to solve problems in areas where computers have previously failed, or have never been tried. Although the literature on expert systems contains much on their construction, on knowledge representation techniques, and programming language developments, relatively little has been devoted to discussing their current application in work-a-day environments. This last point represents the main thrust of this thesis, but some discussion will be devoted to expert system

technology in general.

2.3 WHEN IS A SYSTEM AN EXPERT SYSTEM?

Developments in AI have been made possible by advances in the mathematical disciplines of symbolic logic, probability, and value theory. These, taken together, led to the development in the 1970s of systems which could offer intelligent behaviour in a limited field of expertise, i.e. the development of expert systems.

Different workers use differing definitions of what constitutes an expert system. Four "schools of thought" can be identified.

The pragmatic school considers any system which can reach decisions which previously required human expertise to be an expert system. Thus algorithmic systems (i.e. systems which are essentially computerised flowcharts) and statistical systems (which compute the odds of a decision being correct, given certain a priori information) are both considered to be expert systems.

A second school considers that, in addition to the above, the ability of the system to explain its behaviour and to operate under conditions of uncertainty to be a sine qua non of expert systems.

Other researchers define expert systems as the implementation on computer of Church's Lambda Calculus, Predicate Calculus, Shafer's work on Belief Functions and Zadeh's theory of Fuzzy Logic. This school would class statistical or algorithmic programs as computer aids, in the same category as spreadsheets or wordprocessors.

Cognitive purists argue against this, claiming that expert systems must simulate human behaviour in the domain of expertise.

In this thesis I accept the broad, pragmatic, definition of an expert system.

2.4 A CRITIQUE OF DIFFERENT TYPES OF EXPERT SYSTEMS.

The different types of expert systems each have inherent advantages and disadvantages. Three common methods used to implement expert systems are statistical, algorithmic and rule based structures.

2.4.1 Statistical systems.

Statistical systems have been critisised as being inflexible, of being too simplistic, inapplicable because of insufficient data, having a poor user interface, and incomprehensible to the user (Spiegelhalter 1984a).

Statistical systems use probabilities derived from an analysis of data. Studies quoted by Spiegelhalter and Knill-Jones (1984) indicate that subjective estimates of a priori probabilities are much less accurate that the calculated figures. However, the use of data to develop such a system implies that decisions have to be made in the development stage as to what variables are to be measured. These cannot be modified later without invalidating statistics previously derived. The system also has an extensive appetite for data - a huge sample has to be obtained if acceptable figures are to be derived for rare outcomes or unusual input. Statistical systems are therefore inherently inflexible.

Statistical methods have also been criticised as being too simplistic, as they assume complete distribution independence of

input variables. Statisticians would argue that while statistical systems make this assumption formally and explicitly, rival systems make the same assumption implicitly and in an ad hoc manner (Spiegelhalter and Knill-Jones 1984).

To take an example from weather forecasting, the two rules:

IF rain today THEN rain tomorrow likely,
IF less than 2 hours sunshine today THEN rain tomorrow likely,

might both be true. Together they reinforce the conclusion "RAIN TOMORROW", when rain today, and the number of sunshine hours today, obviously measure related factors.

Proponents of statistical methods also argue that statistical systems are as good as (Spiegelhalter 1984a) or superior to other methods, (Horbar 1985) because statistical systems use more information in reaching a decision. Horbar also argues that semi-quantitative (i.e. fuzzy logic) methods use crudely estimated variables. Errors in these variables may be compounded to unexpectedly large magnitudes and render the model unreliable.

Statistical systems also stand accused of only being able to produce answers. They have no facilities for providing justifications for their answers, and the manner of their operation is opaque to non-statistically minded users. This has been a major criticism of many existing statistical systems, but Critchfield et al (1986) do not see the problem as fundamental. They argue that "explanatory" programs can be written which monitor the functioning of the statistical module, and could provide explanatory functions equivalent to that obtained by other approaches.

2.4.2 Algorithmic systems.

Algorithmic systems, also called "categorical" or "deductive" systems, involve the utilization of a method (or algorithm) for classifying a set of findings as belonging to one of a set of conclusions. These systems do not consider the quantitative balancing of evidence. As real-world problems involve doubt, ignorance, variation and degree of confirmation, such systems are not considered to be expert systems by many researchers.

2.4.3 Rule based systems.

Rule based systems consist of two parts - an inference engine and a knowledge base. The inference engine controls the strategy used to chain through the knowledge base. The knowledge base consists of a series of production rules, which are derived in an ad hoc manner by interviewing experts in the field. This process has been termed knowledge engineering. Although knowledge engineering does not require the sampling data that statistical systems require, the process of writing a knowledge base has proved to be difficult and time-consuming.

Rule based systems have the advantage that prototypes can be developed quickly, and can offer justifications for conclusions by "tracing" the logic that led to that conclusion. As the systems are developed in an ad hoc manner, they can be modified to improve chaining strategies, the user interface or explanatory functions. The knowledge base may also be independent of the inference engine, and so can be modified by people who have little knowledge of computer programming.

The use of fuzzy logic in these systems allows for uncertainty on

the part of the user, and uses quantitative methods for reaching a judgement. Fuzzy logic has been criticised because it defines uncertainty in terms of ad-hoc numbers. The meaning of these numbers to the systems is not always clear, although the numbers may be readily comprehensible to the user.

Consider the example:
If X and Y then Z (certainty factor = 0.7).

It is relatively easy to explain the meaning of this by paraphrasing "If X is present and Y is present then we are fairly, but not absolutely, certain, that Z is also present." However, this begs that question of how this differs from a certainty factor of, say, 0.8. Would the system behave differently in the latter case?

If the certainty factor is a statistic (i.e. in n cases of X and Y both being present, Z would also be present in 0.7 x n cases) then it is simply a crude guess at a probability. If the certainty factor is not a probability, then does it have any actual meaning? Spiegelhalter and Knill-Jones (1984) pose the question "Are we trying to model the clinician's opinion with uninterpretable numbers?"

More sophisticated models elaborate on the simple rule structure "If X then Y" by introducing connections other than "then", e.g. is-caused-by, results-in, side-effect-of, etc. The simple nodes X and Y can also be elaborated into "frames", which store information and connections of their own. Frames can also be located in an hierarchy with inheritance characteristics, e.g. properties of frames high in the hierarchy are also shared by the frames below them, without this having to be explicitly

expressed.

Rule based systems differ fundamentally from algorithmic systems. In algorithmic systems, connections are made explicit by the coding, and all connections exist during execution, even if they are not used. In rule based systems, connections are made during execution, and which connections are made is dependent on the data available.

2.5 NEURAL NETWORKS - A NEW APPROACH TO COMPUTERS.

In the fields of pattern recognition, language comprehension and combinatorial optimisation, the human brain easily out-performs today's fastest digital computers. This superiority is due to the architecture of the human brain. Many billions of neurons, with many more billions of interconnections, operate in a way that is fundamentally different from today's serial computers.

Neural networks, which use structures simulating the connections of neurons in the brain try to emulate the way information processing occurs in living brains, were first researched in the 1950s. After a short burst of excitement, the work which was begun in the 1950s was abandoned (Naylor 1983).

Three developments have rekindled interest in neural networks.

a) Computers are now powerful enough to simulate non-trivial neural network models.
b) The Soviet mathematician Kolmogorov has pioneered formal analysis of neural network structures. He has developed general equations which govern the behaviour of neural networks. His work also proves that neural network models can learn to approximate any continuous mapping function, and

minimise least mean square error.

c) The emergence of new hardware developments, particularly parallel-machine architecture, encourages research into programming paradigms which can take advantage of the new hardware.

A neural network consists of an array of elements (neurons) with interconnections between them, and an Input/Output scheme. Neural networks are characterised by two criteria.

a) The topology of a neural network refers to how many interconnections exist between neurons, and whether these interconnections are between near neurons, distant neurons or some combination of near and distant neurons.

b) Learning rules determine how each neuron interprets information coming from all neurons connected to it, and what signal the neuron will distribute to neurons to which it is connected.

Experimental models of neural networks have demonstrated that neural networks display the basic quality of self-organisation. From this quality other computational properties have emerged, such as association, optimisation and fault tolerance.

An associative memory model is described by Josin (1987). The model consists of 72 neural elements. An 8 x 8 matrix of Boolean elements, simulating lights, is set up. Forty patterns of lights are taught to the neural network. Each pattern in the forty is characterised as belonging to one of eight classes. The patterns are simple, e.g. diagonal line of "on" lights from left top to right bottom of the matrix.

The neural network was then confronted with new patterns, and

asked to classify the new pattern as one of the eight classes. Where patterns were ambiguous, e.g. comprising of 60% of a pattern from one class and 40% of a pattern from a second class, the system detected the dominant pattern. When the split was exactly 50-50 then the system registered "confusion" and declined to classify the pattern. This simple experiment implies that neural networks can adapt to unpredictable changes and can deal with inaccurate input.

A classic optimisation problem concerns a situation of N positions on a map. A closed path must be drawn which visits each position in turn with the shortest possible length. As there are N factorial possible solutions to the problem, it is not possible to calculate every solution and then choose the shortest if N is even moderately large. Josin (1987) reports that a neural network has proved adept at arriving at a near-best solution in a very short period of time.

As the processing power and data storage in neural networks is distributed throughout the network, it is possible to destroy a percentage of the neurons without grossly affecting the behaviour of the system. This implies that neural network architecture will have an inherent capacity for fault tolerance not found in existing computers.

Work on neural networks is being undertaken in many centres throughout the world, including the Institute of Theoretical Physics and Astrophysics at this university (U.C.T.). Commercial neural network simulation programs have recently become available for personal computers.

"MacBrain" is a 200 neuron, 40 000 connection neural network

available for the Apple MacIntosh. NeuralWare Inc. offers a 4000 neuron, 16 000 connection neural network for the IBM-PC, and a slightly smaller system which finds the nearest match to an input word is also currently available for the IBM-PC. All these systems cost under $100.00 in 1987.

A coprocessor board for the IBM PC/AT, developed by Anza Inc., offers a much bigger system, with 30 000 neurons and 480 000 connections, capable of making 25 000 connections a second during learning.

The human brain is estimated to contain some 10^{14} connections. If we accept that the number of connections possible is a measure of processing power, there already exists an artificial neural network of one billionth the power of the human brain.

It is interesting, if facile, to note that computer power is expected to increase by a factor of two every two years. Therefore

$$\text{If } 10^{14} \text{ connections} = (480\ 000 \text{ connections})(2^{x})$$
$$\text{then } x \log 2 = 14 - (6 + \log 0.48)$$
$$\text{implies} \quad x = (8 - \log 0.48) / 0.301$$
$$= 27.6 \text{ two year periods.}$$

This indicates that an electronic (or photronic) neural network on a personal computer, matching the processing power of the human brain will be made available in the year 2043. If the speed of learning of neural networks proceeds at an equivalent rate, the time taken to teach a neural network with human-level cognition ability to become a useful member of society would be in the order of five seconds.

While these figures make interesting (and alarming) reading,

neural network technology is still in its infancy, and obstacles may be encountered which prevent neural networks from being used to solve real world problems. It is nevertheless necessary to realise that computer science is a fast moving discipline, and new developments may render existing ideas, hardware and software obsolete.

CHAPTER 3

EXPERT SYSTEMS IN MEDICINE

"Why can't a doctor be more like a computer?", asks a writer in 'The Economist' magazine (Anonymous 1984).. Rennels and Shortliffe (1987) state "we think the time is not far off when physicians will consider the computer to be as essential a medical instrument as the stethoscope."

Health care is becoming more complex and expensive, and existing resources are insufficient to cope with present demand. Many researchers have looked to computers which can simulate the reasoning process of clinicians as a possible solution. The concept was first proposed in 1959. By 1984 over 1000 papers, describing dozens, perhaps hundreds of such systems, had been published (Spiegelhalter 1984a).

3.1 HISTORICAL OVERVIEW.

Many expert systems have been developed for a variety of clinical problems. Programs written in the 1970s were large-scale experimental prototypes designed to offer diagnostic advice in a limited field (Weiss and Galen 1986). MYCIN, CASNET and INTERNIST (later called CADUCEUS) are the most well-known and influential of these.

The techniques pioneered in the 1970s were utilised to develop many smaller systems covering a broad range of medical problems. Of these, only a handful are used in routine practice.

Systems to interpret electrocardiographs have achieved the widest

acceptance, but have developed independently of the mainstream of medical expert systems. They are discussed in the following chapter. Only four other systems for clinical applications have achieved routine use. ONCOCIN deals with chemotherapy protocols, HELP tracks and analyses a medical record system, PUFF analyses pulmonary function tests and SPE analyses serum protein electrophoresis results. HELP and ONCOCIN offer critiques of a described plan of therapy. SPE-EXPERT, PUFF and electrocardiograph interpretation systems are examples of systems which interpret instrument-based tests.

3.2 LARGE SCALE PROTOTYPES.

3.2.1 MYCIN

The MYCIN project (the name reminiscent of a class of antimicrobial agents, e.g. neomycin) began in the early 1970s, and terminated in 1978. The project represents a milestone in the development of expert systems. Before MYCIN, AI research was concerned mainly with "toy problems" which illustrated theoretical principles. MYCIN was one of the first attempts to address a real-world problem. MYCIN is designed to offer consultative advice on the diagnosis and management of bacteremia and meningitis. (see Coombs 1984, Harmon and King 1985).

MYCIN pioneered the concept of dividing the expert system into two components: an inference engine and a knowledge base. The MYCIN inference engine operates on a backward chaining strategy. Fuzzy logic facilities are offered. Thus rules in the knowledge base are characterised by a certainty factor between 1 and 0. A certainty factor of 1 represents absolute certainty, a factor of zero implies there is no relationship between arguments. Input

variables are characterised by a certainty factor between -1 and 1. A value of 1 implies the feature is definitely present, -1 that it is definitely absent, and 0 that it is impossible to determine whether a feature is present or absent. It is also possible to enter a blank value. This means that information on the variable is not available at the moment.

MYCIN analyses a problem in two phases. A diagnosis is reached first and then a treatment regime is suggested. The MYCIN knowledge base was constructed with the aid of experts in the field, and consists of approximately 600 rules. (Harmon and King 1985).

The MYCIN method of knowledge representation offers some explanation facilities. An input of "Why?" reveals a trace of the reasoning chain and the current tentative diagnosis. However, the system cannot justify the rules themselves.

Studies suggest that MYCIN performs well. Miller (1986) considers that MYCIN has indicated correct treatment and diagnosis in 70% of the cases it has been tested on. Spiegelhausen (1984a) refers to two studies, where MYCIN's recommendations were unacceptable in 27% and 35% of cases respectively. In the latter study, however, MYCIN's performance was better than that of any of the panel of physicians.

The MYCIN research team did not only concentrate on performance. A modified version of MYCIN, called TEIRESIAS, (named after the figure from Greek mythology, who had the power of prophesy), was developed to offer improved explanatory functions.

TEIRESIAS has two types of rules: the basic rules, as in MYCIN,

and meta-rules. TEIRESIAS divides the MYCIN knowledge base into 30 "tasks" by means of 74 meta-rules. The meta-rules are written to guide the inference engine in a manner which is more easily comprehended by the user, i.e. they improve explanatory functions, rather than performance. GUIDON (from the verb "Guide"), another MYCIN derivative, is used for educational purposes. GUIDON contains the inference engine and knowledge base of MYCIN, as well as a database of case studies. When a student approaches GUIDON with a request to study a type of clinical problem, GUIDON selects a suitable case from the database and "solves" the problem exactly as MYCIN would. The student interacts with a second inference engine, which determines whether or not the student's management suggestions are acceptable.

EMYCIN, the inference engine developed for MYCIN, has been released separately as an expert system shell. It has been used with other knowledge bases in a wide variety of medical and non-medical fields.

3.2.2 INTERNIST

INTERNIST (reflecting the desire to cover the whole of internal medicine, a name changed later to CADUCEUS) represents what is perhaps the most ambitious project in medical AI to date. It was begun in the early 1970s, before the MYCIN project, and the first version was demonstrated in 1974.

INTERNIST claims to be an expert on 70-75% of the entire field of internal medicine (Torasso 1985). The knowledge base took 15 person-years to produce. It consists of 500 profiles with 3550 manifestations of disease. There are 2600 links between diseases,

6500 between manifestations and 100 000 other associations. (Miller et al. 1982).

INTERNIST stores its knowledge in "frames". A frame consists of a diagnosis (disease or condition) and a disease "profile". Each profile consists of between 25 and 200 findings - signs, symptoms and laboratory results. Each finding is associated with two variables, the "frequency" and the "evoking strength".

"Frequency" relates to the prevalence of a finding in patients suffering from the disease, and is an integer value from 1 to 5. A value of 5 indicates that the finding is present in practically all patients who have the disease. A value of 1 indicates that the finding can occur in patients with the disease, but rarely does.

"Evoking strength" is an integer between 0 and 5 inclusive, and refers to the specificity of a finding. A value of 5 indicates that the finding is pathognomonic of a particular disease, e.g. MALARIA PARASITES IN BLOOD is pathognomonic of malaria. A value of 0 indicates a nonspecific finding, e.g. FEVER, MALAISE.

Disease profiles were originally arranged in an hierarchical structure, where findings in a disease profile high in the hierarchy also applied to profiles below it. However, this system was found to be inadequate as certain diseases could be classified in more than one place in the hierarchy. The schema was abandoned in favour of a more ad hoc approach.

"Property lists" exist separately from the frames. The "property lists" define a list of "properties" for every sign, symptom or laboratory test referred to in the knowledge base. Fourteen kinds of "properties" exist. The "Import" property is an integer which

describes how important it is to explain the findings. Findings with a low "Import" can be ignored if they do not correspond with the selected diagnosis, and diagnoses can be ignored if they do not explain findings with a high "import". Other properties define populations in which a finding can or cannot occur, e.g. men cannot be pregnant. Properties also group findings with other findings e.g. one liver function test would be "grouped" with other findings about the liver, and contain various types of logical connections between interdependent findings.

INTERNIST works through the knowledge base by first asking questions that would achieve a spread of information across all the disease profiles. When one disease profile emerges as a strong candidate, questions are directed to confirm or exclude that disease.

"Properties" in INTERNIST are analogous to the rules in MYCIN, and help direct INTERNIST's requests for information. Frames (disease profiles) can be considered as extensions of the MYCIN decision node concept.

INTERNIST was developed as a research project, and was never intended for clinical use. A study based on the notoriously difficult case studies published in the New England Journal of Medicine showed that INTERNIST functioned well when the diseases in question were covered by the knowledge base, which occurred in about 50% of the cases.

In the remaining cases, failure of INTERNIST to perform correctly was attributed to incorrect evoking strength, frequency or import figures, and the lack of adequate anatomical knowledge. INTERNIST was also criticised for a lack of temporal reasoning. In

addition, it was unable to form an overview of complex problems because of its ad hoc classification system, and thus could not attribute findings to proper causes.

Rennels and Shortliffe (1987) describe a microcomputer version of INTERNIST called QMR (Quick Medical Reference) which functions as an electronic textbook.

3.2.3 CASNET

CASNET (Causal ASsociation NETwork) adopts a representation which stresses the causal relationship in the knowledge base. The nodes in the network represent findings or hypotheses, and links between the nodes represent relationships.

CASNET requires the knowledge base to be written in the form of rules. It supports fuzzy logic, and adopts a forward chaining strategy. CASNET was developed in conjunction with a knowledge base for the diagnosis of glaucoma. It was later released as an expert system shell under the name EXPERT. EXPERT has been used to develop many systems, including medical consultation models in dermatology, rheumatology and haematology. (Quaglini and Stefanelli 1986).

3.3 SMALLER PROJECTS.

While MYCIN, INTERNIST and CASNET were successful in demonstrating that AI research could be used to solve practical problems, none of them has proved useful in a routine clinical setting. Some smaller projects, building on the technology they developed, have been used in a routine environment.

3.3.1 ONCOCIN

ONCOCIN (from "oncology", the study of cancer) is, as its name suggests, a progeny of the MYCIN project. It is designed to assist in chemotherapy for cancer patients. As chemotherapy is a very hazardous form of treatment, patients undergoing chemotherapy are treated according to strict guidelines, called protocols. It is important for both therapeutic and research reasons for cases to be carefully followed up, and detailed records and progress reports kept.

ONCOCIN functions both as a medical record and as an advice system. The medical record module replaces the filling out of patient data forms, and also prints progress reports previously filled out by hand. This saves the doctor time, and in this system, played a large part in overcoming the clinician's resistance to computerisation.

The advice system utilises the information provided to the medical record module. The advice system initially suggested a strategy for further treatment. Physicians using the system found that the suggested strategy often required modifications, and that it was annoying and time-consuming to enter the modifications. The system was therefore changed. The physician would first enter his or her own plan of therapy. The system would compare this with its own decisions, and comment only if major differences existed. (Coombs 1984, Rennels and Shortliffe 1987).

The first version of the system was demonstrated in 1981. Clinical efficacy trials have shown that better records and more consistent treatment regimes result from using ONCOCIN than from

manual systems. (Miller 1986).

3.3.2 HELP

HELP follows the critiqueing model pioneered by ONCOCIN, but is a much larger system. It has been adopted in at least three American hospitals, and is commercially available for $1.5m. in 1984 (Anonymous 1984).

HELP incorporates a complete hospital information system, and monitors the medical records entered. HELP utilises conditional probabilities for conditions, and rules for others, e.g. drug interaction and normal values of laboratory tests. When a potentially dangerous situation is detected, an "Alert" message is issued. Studies have shown that 1.8% of all hospital admissions have generated Alert signals, and that action is taken more rapidly when the HELP system is operational.

3.3.3 PUFF

PUFF is designed to interpret the results of lung function tests. The system interfaces directly with the lung function equipment, and so requires minimal user interaction. It was installed for routine use in a lung function laboratory in 1979. A study of over 4000 cases showed that 95% of reports generated by PUFF were accepted by the pulmonary physician without any modification (Blum 1985).

PUFF was developed to run under the EMYCIN expert system shell, but has since been converted to BASIC. The knowledge base initially had 64 rules concerning 54 parameters, but this has been expanded to 400 rules and 75 parameters. The system offers only rudimentary explanatory facilities, but this has not been

found to be a drawback. One reason suggested for the acceptance of the PUFF system is that the staff of the Lung Function Laboratory were already familiar with computers (Harmon and King 1985).

3.3.4 SPE-EXPERT

SPE-EXPERT (Serum Protein Electrophoresis, using the EXPERT shell) is a fairly small expert system, designed to interpret serum electrophoresis results. It consists of 107 rules, requires 25 parameters, and offers 33 interpretations. The original knowledge base was written for the EXPERT expert system shell, and ran on an IBM-PC interfaced directly to the electrophoresis equipment. A later version was converted to machine code, and the whole system was incorporated into the instrument. The system requires no user interaction, and has no explanatory facilities (Weiss and Galen 1986).

A study of 256 samples, quoted by Harmon and King (1985) concluded that SPE-EXPERT was correct in its interpretation in 100% of the sample.

3.4 CONCLUSION.

The large scale experimental expert systems developed in the 1970s demonstrated that the mathematics of symbolic logic, and programming languages such as LISP, PROLOG and SMALLTALK, could be utilised to address real world problems. However, the prototype systems were not precursors of production systems. Large scale systems were abandoned in the 1980s in favour of the development of small useful systems. Of the scores, perhaps hundreds, of systems that have been developed only a handful are

used in routine practice.

Adlassnig and Kolarz (1986) feel that expert systems should not be a substitute for human ability, but extend human expertise. In some fields this has happened. PROSPECTOR, a geology expert system, is reported to have discovered new molybdenum deposits in the United States, and EURISKO is credited with inventing a new AND/OR gate in integrated circuit design (Forsyth and Nalor 1985). In the medical field, however, a substantial amount of pioneering research has resulted in a dismal number of routinely utilised systems, and all of these are used essentially as screening tools which require expert supervision.

Simple prejudice on the part of the medical profession has been blamed for the failure. "The autonomy of senior clinicians," concludes Spiegelhausen (1984a), "mitigates against procedures that tend to impose a degree of publicness (sic) and consensus-seeking on clinical practice. Given entrenched attitudes and beliefs, formal attempts (to do this) are inevitably doomed to failure." Harmon and King (1985) list the doctor's pride and inertia, as well as the disruption of the doctor's routine, among reasons for the poor acceptance of medical expert systems.

Rennels and Shortliffe (1987) agree that clinicians would reject a system that gave insufficient explanation, even if it had good diagnostic accuracy. They feel that the provision of excellent explanatory functions is the key to the acceptance of medical expert systems. Harmon and King also conclude that medical expert systems require friendlier interfaces and better explanatory facilities.

Torasso (1985) considers that more work is required to clarify

areas where automatic decision assistance can be useful and convenient. He suggests medical education as one likely area. Torasso also notes that major problems exist with knowledge acquisition and validation. There is often little agreement among experts in a field. Physicians working in different geographical areas often consider different clinical data. There is considerable observer variability among physicians, and diseases present differently in different populations. All these factors mitigate against successfully transferring expert systems from one site to another.

Systems which are in routine use are either essentially extensions of instruments (e.g. PUFF, SPE-EXPERT, electrocardiograph interpretation) or extensions of computerised record systems (HELP, ONCOCIN). That these systems can be considered as extensions of already familiar equipment may play a role in overcoming prejudice against their use. As the systems interface to other equipment, the need for excellent user interfaces does not arise. Explanatory functions in these systems are either absent or rudimentary. In these instances explanatory functions may not be crucial. Physicians are familiar with accepting reports on a great variety of special investigations, from biochemistry tests to radiographic studies.

From this discussion, it seems likely that expert systems in medicine could be most easily incorporated in "smart" instruments. The development of explanatory functions, particularly those suitable for use in medical education, is also identified as a priority.

Hasling and co-workers, quoted by Coombs (1984), identify three

types of explanatory functions.

a) Epistemologic functions concern the structures used to represent knowledge and reasoning in the expert system. Tracing the system's route through the structure provides a justification for the program's actions.

b) Rhetoric functions are concerned with stating the explanation so that it can be understood. This entails dealing with problems of English grammar and syntax, and the use of other methods such as graphics and animated graphics.

c) User model functions. Human experts are generally called upon to provide conception guidance to other experts, not to act as "wise men". A system must therefore generate explanations that take user knowledge and preference into account. A complete novice will need step-by-step guidance in a subject. A more experienced user might find the "critiqueing model" more acceptable.

Computers have, on average, doubled in power and memory capacity, as well as halved in price, every two years since 1948. This trend shows no sign of slowing down. This dynamic alone insists that computers will play a greater and greater role in medical practice. However, unless unexpected technological breakthroughs are made in expert system technology, the development of medical expert systems will be a slow and incremental process. The design of "smart" instruments and the provision of explanatory and teaching functions appears to be a realistic goal for software developers.

CHAPTER 4

A REVIEW OF COMPUTER INTERPRETATION OF ECGS

The concept of automated ECG interpretation was first proposed in 1957 (Pryor et al. 1980). The first functional systems were available in the early 1960s, the first comparisons of computerised versus physician interpretation were done in the middle 1960s, and the first routine computerised ECG services were available in the late 1960s. Since then many commercial systems, for both mainframe based systems, and more recently, for microprocessor based systems have been developed.

Existing systems for the interpretation of ECGs are reviewed in order to obtain guidelines for the development of the ECG Tutor. An historical survey gives an indication of how ambitious the task of interpreting ECGs by computer actually is. The historical survey also suggests areas where existing research and development can be utilised in the proposed Tutor, and how much of this is freely available in the public domain. The internal structure of a typical system for interpreting ECGs is discussed, and the problems inherent to such systems highlighted. Studies which evaluate the performance of ECG interpretation by computer are also considered.

4.1 HISTORICAL SURVEY.

4.1.1 Early Systems

The first project to develop a computer program to interpret ECGs was the American Veterans' Administration program, begun in 1957 by Pipberger and co-workers (Pryor et al. 1980). This system

adopted a Bayesian statistical method for reaching interpretations, and analyzed leads of the Frank (also called orthogonal or XYZ) system. Although developed and improved over a period of ten years, neither the Bayesian statistical method nor the Frank lead system gained widespread acceptance. The inherent limitations of Bayesian statistical methods have been discussed in the previous chapter. The XYZ lead system is considered by many cardiologists to be inferior to the 12-lead system because more information is present in the latter system. Most interpretative ECG systems have followed the ECAN model.

Development of the ECAN system (ElectroCardiograph ANalysis) began in 1959 with sponsorship from the United States Public Health Services. The Federal U.S. Government invested $11 million in the project over the next twelve years (Caceres 1976). This figure gives some indication of the difficulty of producing a computer system to interpret ECGs. The latest version, ECAN-E, was released in the public domain in 1973.

ECAN is written in FORTRAN and assembler. It utilises the standard twelve lead ECG and a knowledge base similar to (but pre-dating) that developed in the MYCIN project.

Phone-a-gram Inc. and Telemed Inc. adopted the ECAN program, and with slight modifications offered an ECG interpretation service where users could access the program, running on the suppliers' mainframe computer, via telephone lines.

Telemed further modified the ECAN program before developing their own software. Over a thousand sites in the U.S.A. were using the Telemed service routinely in 1976 (Pryor et al. 1980).

IBM developed their own program under the direction of Dr. Raymond Bonner in the early 1970s. IBM did not offer an ECG interpretation service, but allowed third party vendors to operate their program under licence. Hewlett-Packard entered the field in 1971, first with the ECAN program, and then offered the IBM/Bonner program. Their own program, which had been under development since 1968, became available in 1977, and is now widely used.

Many other systems were also developed both in the U.S.A. and in Europe, e.g. the Cro-Med system developed at Mt. Sinai Hospital, (used at 125 sites in 1980) and the Mayo system developed at the Mayo Clinic (released in 1976). Many of these are in the public domain, but most of those used at all in routine practice were only used at the original site of development. Almost all of the later projects in interpretative electrocardiography began by modifying an existing system.

Dudeck (1980) estimates that 10 million ECGs were analysed by computer in the U.S.A. in 1979, and Caceres (1976) suggested that the total investment in computerised ECG interpretation between 1959 and 1975 (including development costs and hardware) was in the region of $350 million. In 1977 over seventy percent of all computer interpreted ECGs were interpreted on the Telemed or IBM systems (Goetowski 1977), but Hewlett-Packard has since made a significant impact on the market. Table 4.1 lists some of the available early programs. While more development work has been done since the 1970s to improve and modify ECG analysis systems, later programs are based on the approach pioneered in the 1970s.

PROGRAM	AVAILABILITY
ECAN	Public domain
PHONE-A-GRAM	Service fee
IBM (Bonner)	License
HEWLETT-PACKARD	Dedicated
CRO-MED	Service fee
TELEMED	Service fee
AMERICAN VETERANS ADMINISTRATION	Public domain
MAYO	Service fee
LDS (Pryor)	Public domain

Table 4.1 : The availability of various ECG analysis programs.

Computer systems for the interpretation of ECGs are more widely used, and at a more advanced stage of development, than any other expert system with medical applications. This not because the interpretation of ECGs is in any way simpler or more amenable to computerisation than other tasks. It is a reflection of the considerable amount of well-funded research which has been devoted to the subject.

This has various implications. A considerable amount of the work that has been done is in the public domain, and can be used as a starting point for any future developmental project. The corollary to this is that any future project has to be superior to the work already completed. It is thus necessary to have access to existing software and a testing strategy to establish whether a novel approach to the subject is an improvement on existing approaches. This applies to the system as a whole, and to areas such as the knowledge base and pattern-recognition algorithm.

4.1.2 Hardware requirements

The technology, developed and implemented for computer interpretation of ECGs on a commercial scale in the early 70s, was obviously dependent on the hardware readily available at that time. Figures for memory size of the mainframe computers used are given in table 4.2.

Computer	Memory
IBM 360/50	512kb
IBM 1800	32k word (64kb)
HP 2100	64kb
PDP 11/35	32kb

Table 4.2. Memory available on some of the computers used for interpretative electrocardiography.

Today's personal computers have the equivalent power and over ten times the amount of memory compared to the mainframes of the early 1970s.

The systems of the 1970s were written in either assembler or FORTRAN IV, or both. Some programs (e.g. the IBM program) which were originally written in assembler were later translated into FORTRAN IV to overcome the problem of portability.

Interpretative electrocardiography programs require about 300kb to 500kb of memory (Poppl 1980). As this amount of memory was beyond the capabilities of many the computers of the period, it was necessary to devise complicated systems of overlays to get the programs to run on the available hardware. This resulted in loss of speed in the system. Computation time for the analysis of the ECG varied from 40 seconds to five minutes, depending on the

system (Pryor et al 1980).

Vendors of interpretative electrocardiographs have not ignored developments in computer hardware. Hewlett-Packard (1985) have ported their FORTRAN interpretative ECG program from the HP-2100 onto a 63000 microprocessor utilising microprogramming techniques. The the 63000 system has been incorporated into the casing of a standard ECG amplifier, and runs in near real time. Other vendors have produced similar systems.

This discussion implies that a personal computer is capable of supporting a program which interprets ECGs.

4.1.3 Installation description.

Early systems were run on mainframe computers. These were located at a central processing centre, and the ECG signal transmitted over telephone lines to the computer. The company operating the system provided the service of an experienced electrocardiologist to overread abnormal ECGs . The computer report with the overreading was then forwarded to the physician who requested the service. Many larger hospitals, including some in South Africa, have a in-house system to service the hospital (see Figure 4.1 for a flowchart of such a system).

Microprocessor based systems allow both the recording of the ECG and the computer interpretation to be done at the bedside. While this has cut costs dramatically and provides an almost instantaneous computer report, some authors (e.g. Drazen 1980) consider that the easy availability of systems will tempt end users to use them without expert (cardiologist) review of abnormal ECGs. This can lead to the overdiagnosis and mis-

diagnosis of cardiac pathologies.

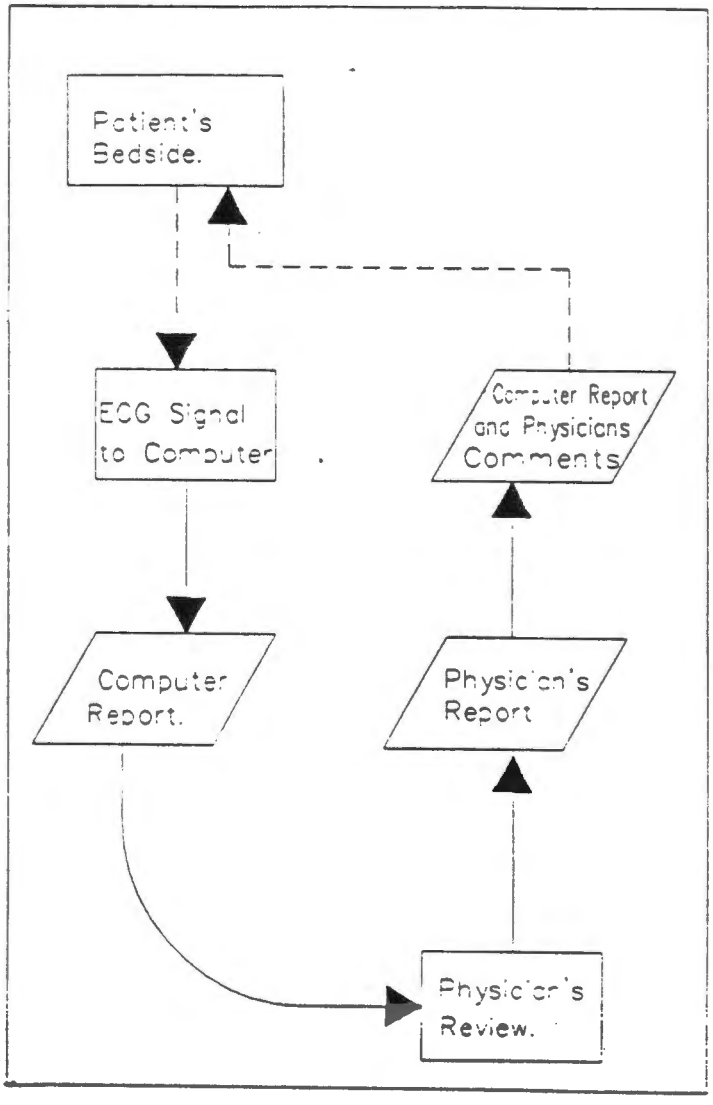

Figure 4.1 Flowchart for the installation of a mainframe based ECG system.

4.2 A DESCRIPTION OF A TYPICAL INTERPRETATIVE ECG PROGRAM.

Most interpretative ECG programs follow the system originally developed for the ECAN project, and this can be considered to be the typical implementation.

4.2.1 Input

An ECG signal is obtained by amplifying the electrical activity recorded from particular points on the body surface. The classical 12-lead convention requires the readings from six sites on the anterior and left side of the chest, and from the limbs. The Frank (also called the XYZ or vector system) convention requires readings from eight sites on the body surface, three of which are roughly orthogonal in relation to the heart. Many other lead arrangements have been proposed. Body surface mapping, which requires a large number of leads, sometimes hundreds, to obtain a contour map of the electrical potential over the entire torso, is currently attracting much attention.

The ECG signal is digitised and fed into the computer. Other parameters, e.g. the age and sex of the patient, must be entered from a keyboard or other device.

4.2.2 Pattern Recognition

The pattern recognition required of ECG analysis systems refers to the process of recognising and identifying the P, QRS and T wave in the ECG trace. The electrical activity recorded on the body surface represents activation of the atria and ventricles, and repolarisation of the ventricles, as atrial repolarisation

co-incides with, and is much smaller than, activity corresponding to ventricular activity. By convention, the deflection representing atrial activation is called the P wave, the deflections representing ventricular activation are referred to as the QRS complex, and the deflection representing ventricular repolarisation are refered to as the T wave.

Leads with low amplitutes pose difficulties for pattern recognition because the signal/noise level becomes unacceptably high. Low voltages are common when the lead direction is approximately at right angles to the so-called resultant cardiac electrical vector. This problem can be avoided to some extent by considering three simultaneous leads. As the leads all "see" the cardiac vector from different angles, is unlikely that low voltages will be present in more than one of three simultaneous leads. If cost is no object, a case can be made for the recording of all twelve leads simultaneously.

In normal rhythm, contraction of the atria precedes the contraction of the ventricles, and this is reflected in the ECG by the P wave preceding the QRS complex. While the QRS complex has fairly distinctive features, the P wave (and the T wave) are comparatively small and bland deflections. The simplest method of identifying these is to first identify the QRS complex, and then label deflections before and after the QRS complex the P and T waves respectively. This algorithm is only valid for normal rhythm.

In arrhythmias, the P wave may or may not precede the QRS complex. It is therefore necessary to first determine whether or not normal rhythm is present.

Detection of arrhythmias has received considerable attention in the development of automated intensive care monitoring equipment. Algorithms employing advanced logical and statistical procedures are discussed in texts on the subject. Algorithms for use in interpretative electrocardiology can be much simpler as continuous monitoring in real time is not required. Broadly speaking, they detect whether the QRS complexes occur regularly, and within the range of normal frequency. Allowance must also be made for detection of ectopic (i.e. occasional) beats and dropped (i.e. occasional missing) beats.

Some systems utilise a separate rhythm strip for rhythm analysis, which can be anything from 8 seconds to 60 seconds long, depending on the system (Pryor et al. 1980). Other systems use the same signal for both rhythm and morphological analysis.

Many pattern recognition algorithms have been described (see Yu et al. 1985; Talmon and Hasman 1980; Pipberger 1977, and Doue and Valance 1985). The large number of algorithms available reflects the dissatisfaction which exists among researchers concerning the performance of the algorithms. While most perform well on normal ECGs, the presence of unusual ECG wave shapes and the presence of noise and artifacts often lead to incorrect wave identifications. Figures for the performance of a pattern-recognition algorithm are dependent on the quality and type of ECGs selected for the data sample, and are therefore difficult to interpret. Talmon and Hasman (1980) felt that there was little to be gained in choosing between the alternate pattern-recognition algorithms then on offer, but one of the most widely used algorithm (Doue and Vallance 1985) was developed since Talmon did his evaluation.

An adequate discussion of the merits and demerits of all pattern

recognition algorithms that have been devised is well beyond the scope of this thesis. However, two algorithms (one early, one developed recently) are discussed in detail, and give some indication of the magnitude and complexity of the problem.

One of the earliest algorithms was developed by Pipberger for the AVA program in the early 1960s. Pipberger (1977) uses the Frank (XYZ) lead convention, and uses the concept of a standard spatial velocity curve to identify the waveforms of the ECG. The spatial standard velocity curve was derived during program development by

a) Collecting a sample of a few hundred ECGs in digital form.
b) Visually identifying the beginning and end of each QRS complex in every ECG in the sample.
c) Computing the smoothed derivatives x'(t), y'(t) and x'(t) from the digitised leads x(t), y(t) and z(t).
d) Computing the velocity curve V_t according to the expression

$$v_t = \text{square root} (x'^2_t + y'^2_t + z'^2_t).$$

The points identifying the beginning and end of the QRS complex (derived in (b)) are superimposed on the velocity curve.
e) The standard spatial velocity curve $V(std)_t$ is obtained by averaging the velocity curves, with the identification points superimposed, for all ECGs in the sample.

During program execution, the leads are digitised and the beginning and end of the QRS complexes identified by

a) Computing the smoothed derivatives x'(t), y'(t) and x'(t) from the leads x(t), y(t) and z(t).
b) Deriving the spatial velocity curve v_t, such that

$$v_t = \text{square root } (x'^2_t + y'^2_t + z'^2_t).$$

c) Defining the onset point of the QRS complex by minimising the least squares error between the standard velocity curve $V(std)_t$ and the calculated v_t, i.e. minimising the expression

$$\sum_{i=1}^{3} ((V_{(t-tq)} - V(std)_t)^2 / w_t)$$

where

tq is the onset (or end) point of the QRS complex, and

w_t is the variance of the standard velocity curve $V(std)_t$.

d) The procedure is repeated to find the end of the QRS complex.

e) The P-wave is identified by locating the first deflection before the QRS complex which exceeds a spatial velocity threshold.

g) The T-wave is identified by searching the region after the QRS complex.

This algorithm has the drawbacks of being a) dependent on the data selected in deriving the standard spatial velocity curve, and b) the subjective nature of the visual coding of that data. The Hewlett-Packard algorithm avoids these pitfalls, and is described below.

The Hewlett-Packard algorithm:-

a) Computes a weighted sum of the first and second differentials of three simultaneous digitised leads to generate the waveform boundary indicator (WBI). Using three simultaneous leads compensates for possible low voltages in one lead.

b) The WBI has empirically been shown to be maximal over the QRS complex region, but slightly narrower than the actual QRS complex. To compensate for this the raw ECG data is convolved

with an impulse response. The impulse response is a triangular waveform, maximal over high points in the WBI. A new WBI (WBIF) is then calculated.

c) The WBIF indicates the QRS search region. The QRS complexes are identified. The S-T segment, the T wave and the P waves are identified in a manner similar to the Pipberger algorithm.

4.2.3 Measurement

Once the P, QRS and T waves are identified, the parameters are measured. The fundamental measurements, i.e. wave amplitude and duration, are indicated in figure 4.2. In addition, about thirty other measurements are taken per lead. These are not shown in the diagram, but include the J points (where the concave deflection in the T wave becomes convex, and visa versa), and the amplitude of the ST segment sixty milliseconds after the end of the S wave.

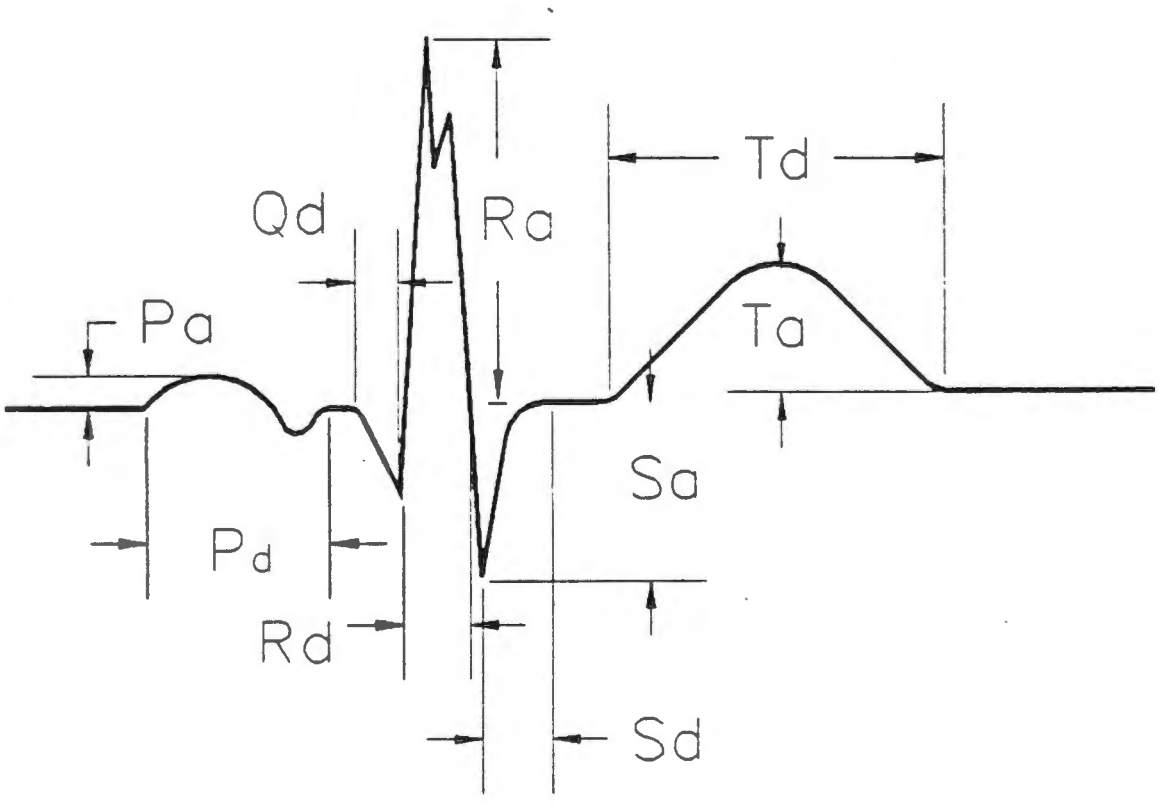

Figure 4.2 : The fundamental measurements of the waveform.

Systems using 12-lead tracings choose measurements which mimic the diagnostic strategy of the cardiologist. Systems using the Frank lead system have more freedom in generating measurements and their criteria for interpretation. Potentially useful parameters not easily identified visually, e.g. velocity vector sums over three leads, have been explored. This method has the disadvantage of not being easily understood by traditional electrocardiologists, and is not widely used.

4.2.4 Interpretation

Once the measurement matrix is obtained and other variables, e.g. patient's age and sex, have been provided, interpretation commences. Two basic strategies have been developed. Bayesian statistics have been utilised to provide interpretations based on tables of prior probabilities. Deterministic, or rule based systems, derive a series of rules from traditional electrocardiology, and chain through these rules to arrive at an interpretation.

The advantage of rule-based systems is that rules derived for one program can easily be applied to another, and it is therefore easier to separate errors of rules from errors of measurement. The logic behind the rules are also readily understood by traditional electrocardiologists. Rules have the disadvantage of being sensitive to cut-off points. i.e. if a rule states "abnormal if greater than 140", values of 139 will be considered normal and values of 141 abnormal, whereas this degree of variability could be obtained in two sequential tracings from the same patient. This problem could be overcome by not using sharp cutoffs, but by "smoothing" the edge by using fuzzy logic.

4.2.5 Comparative functions

Systems for interpretative electrocardiology treat each ECG tracing as a new problem. Studies (Bonner 1983) indicate that many ECGs are not single studies, but are followed up by subsequent tracings. Bonner has developed an ECG program that allows serial comparisons, but the system was not released commercially.

4.3 PROGRAM EVALUATION

No standard system for the evaluation of interpretive ECG systems has been devised. Most designers of ECG programs devise their own tests of diagnostic capability (see Anonymous 1985). Caceres (1976) has noted that the performance statistics for each new program tend to be similar. He recommends that rigorous performance specifications be devised, and suggests further that all reports about computer programs be discontinued until more objective evaluation criteria have been developed. Pryor et al (1980) agrees that most published evaluations are inadequate.

The 1977 Bethesda Conference on Optimal Electrocardiography (Milliken et al 1983) suggested that interpretation statements concerning the ECG be classified as Types A, B, and C. Type A statements refer to the presence or absence of specific diseases or clinical conditions which can be confirmed by non-electrocardiographic methods, e.g. enzyme studies, post-mortems. Type B statements refer to electrocardiographic conditions, e.g. electrical axis, rhythm abnormalities. Type C statements refer to non-specific electrocardiographic changes. While Type A

interpretations can be verified against a known diagnosis, Type B and C statements can only be compared against the opinion of a cardiologist or another program.

Richards et al. (1980) studied Type A statements produced by the IBM/ Bonner program for atrial and ventricular hypertrophy against autopsy evidence. Left ventricular hypertrophy was correctly detected 53.6% of the time, and right ventricular hypertrophy 84.0% of the time. In contrast, left or right atrial enlargement were detected 10.8% and 18.5% of the time respectively. Richards concludes that the accuracy of the program is not at fault - the ECG is simply not a very accurate tool in the detection of these conditions.

Studies of computer versus cardiologist interpretation pose the question "How accurate is the cardiologist's interpretation of the ECG?"

In one study (Bernard et al. 1983) three expert electrocardiographers were asked to code pre-selected ECGs according to the Minnesota classification system. Of the sample provided, all three electro-cardiographers agreed that twenty-two records showed ischaemic heart disease. Two of the three selected a further eight records as having ischaemic heart disease, and another twenty-six records were selected by one or other of the three. This study indicates that even when interpreting ECGs according to predefined criteria, experienced cardiologists differ markedly.

Another study (Milliken et al. 1983) showed that on repeat readings of the same ECGs, nine experienced electrocardiographers made major changes in between 8% and 22% of the sample. The same

study showed that in instances of Type A statements, where the true diagnosis was known, the cardiologists were correct between 46% and 67% of the time, while the AVA computer program was correct 76% of the time.

Bonner et al. (1979) defined a check list of 238 statements which could be made about the ECG, and asked seven cardiologists to interpret 879 ECGs by selecting statements from the list. All seven cardiologists chose the same statements for 17% of the ECGs. Six of the seven agreed on a further 9%, and five of the seven and four of the seven on a further 7% each. In 36% of the sample there was no agreement among the cardiologists.

These studies give the impression that even experienced electrocardiologists differ among themselves, and a single cardiologist may interpret the same tracing differently at different times. It is, however, difficult to evaluate physician performance in interpreting ECGs because physicians also use "soft" criteria, e.g. patient's work, body deformities, in arriving at an interpretation.

Various studies have been done comparing computer performance against the consensus of a panel of physicians. Laks (1980), in a study of 16 000 ECGs taken consecutively between July 1977 and July 1978 found 87% were correctly interpreted by computer. This rose to 99% for normal ECGs. Caceres (1976) quotes figures of 75%-85% correct interpretation for morphological abnormalities, but 66% for rhythm abnormalities. Bonner et al. (1983) considers the interpretation from the IBM/Bonner program to be roughly correct 97% of the time, and completely acceptable 82% of the time. Bernard et al. (1983) and Milliken and Henderson (1978) report similar findings on rival interpretation programs.

Milliken and Henderson (1978) notes that, on average, 33% of the over 1 million ECGs interpreted annually over the previous few years by the Telemed system are rejected as being of inferior technical quality. Once tracings of inferior quality are rejected the ability of the computer system to perform rises rapidly. Pryor et al. (1980) identifies the computer's difficulty in dealing with n in the signal as the overriding factor in the computer's ability to perform adequately. Pryor et al. (1980), Cerutti et al. (1980) and Gold (1985) all argue for preprocessing of the ECG to improve signal quality. A high pass filter will correct baseline drift, and a 50Hz notch filter will remove mains interference, but both can distort the tracing. Standards for ECG amplifiers were originally devised for visual inspection of ECGs, and are not adequate for defining the requirements of amplifiers used with computerised systems. Some widely used amplifiers have been shown to introduce artifacts, non-linear phase response over the required frequency range has been identified as a culprit (Taylor and Vincent 1983, Anonymous 1985).

Computer interpretations are also not precise. Pryor et al. (1980) reports on a study where ECGs were sampled at 1000 Hz. Odd numbered samples were collected for one signal, and even numbered samples for a second signal. This results in two tracings sampled at 500 Hz each. The two tracings thus represented the same ECG with samples taken 1 millisecond apart. The IBM/Bonner program only recognised the tracings as identical in 95% of cases of normal ECGs, and 63.6% of the cases of abnormal ECGs. This may be the result of measurements falling on the border of a criterion, e.g. if a P-R interval of greater than 120 ms is indicative of a condition, a tracing with a P-R interval of 120.1 ms will indicate

the presence of the condition, while a PR interval of 120.0 ms will indicate the absence of the condition.

In signals of adequate technical quality, failure to correctly identify the waveforms is the greatest source of error. Balda et al. (1977) claims the Hewlett Packard program (HP-6) correctly identifies 89% of P waves, 96% of T waves and 98% of QRS complexes. Pryor et al. (1980) notes that over 700 types of rhythm abnormalities have been described, and Caceres (1973) notes computer programs can only deal with a small fraction of the most common of these, generally about thirty.

Talmon and Hasman (1980) feel that there is little to be gained in choosing between the alternate pattern-recognition algorithms now offered, and that we are reaching the limits of technology to extract fast gains from redefining ECG classifications procedures. Pryor concludes that technology still requires that no interpretative ECG should be allowed to operate without routine overview by a human reader. In practice only abnormal traces are routinely overread by a cardiologist (Drazen 1980). In effect, computer interpretative electrocardiology is used as a screening tool, rather than a diagnostic tool.

This distinction is apparent to physicians in hospitals where overreading of ECGs is routinely available. The distinction may not be apparent to the general practitioner who has invested in a bedside ECG amplifier and interpreter.

4.4 ECONOMIC CONSIDERATIONS

Much of the published work on the costs of interpretative

electrocardiology is relevant only to the United States, and much of that is outdated. Drazen (1980) reports the cost of the computer interpretation of the ECG to be between U.S.$1.50 and $4.25 per tracing, depending on volume. Service vendors in the 1970s charged between $5.00 and $7.50 per tracing, plus $3.00 to $5.00 for overreading. This service, including overreading, is above what a private cardiologist would charge at that time. Drazen feels that only abnormal ECGs need to be overread. As abnormal tracings constitute only about 20% of routine tracings, Drazen argues that computer interpretation is cost effective.

Drazen also quotes a microprocessor-based interpretative system as costing $20 000 for hardware, and $350 per month rental for the software. He quotes $8 000 as the cost of a normal ECG cart. Thus $12 000 and $350 per month is the additional cost of the computerised interpretation.

In 1986 Hewlett-Packard were marketing their interpretative cardiograph (Hewlett Packard 1984) at R23 000, including software and service. The standard (non-interpretative) model ECG cart retailed at R20 000.

4.5 FUTURE TRENDS

Programs for interpretative electrocardiology have gone through three stages. The first stage, in the sixties, saw government funded research projects bring a concept to the market place. In the second stage, commercial concerns improved on the programs and offered interpretation services, using centrally-based computers. The third change saw the same programs converted to run on dedicated microprocessors, with a considerable decrease in

cost.

Much current development work is carried out by relatively small concerns using personal computers. Clinical Microdata Systems Inc. (1985) market a system based on the HP 150 personal computer which allows simultaneous processing of the ECG, phonocardiogram and carotid pulse measurements for patient monitoring, and offers limited interpretative functions. Some workers (e.g. Golding 1986) have developed microcomputer-based programs to teach the electrophysiological basis of the ECG and the subtlety of motion of various cardiac arrhythmias, but these programs are of limited capability and run on very small computers. (Golding's program was developed for an Apple II with 64kb of RAM).

It is suggested that the educational functions of the latter programs, as well as the explanatory functions developed for other medical expert systems, are combined with the interpretative functions of an ECG computer analysis system. This combination would produce a system for teaching electrocardiology, but would not be confined to textbook examples. Students could record tracings off their own patients, and gain insight into the interpretation of the ECG and the underlying abnormality. Specifications for such a system are provided in the next chapter.

CHAPTER 5

SPECIFICATIONS FOR THE ELECTROCARDIOGRAPH TUTOR.

5.1 SPECIFICATIONS.

The system is required to do the following

5.1.1 The input module

The input module must:-

i) Accept a 12-lead ECG as the input, either directly from the patient or from a paper record.

ii) Convert the signal to digital form without losing relevant information. The resulting digital signal must be suitable for signal processing to identify the salient features of the waveform.

iii) Identify the salient features of the waveform correctly. The user must be able to visually identify the wave features, and compare these with the computer identification.

iv) Make the measurements of the salient features of the ECG available to the expert system module.

5.1.2 The expert system.

i) The expert system must be able to function with more than one knowledge base. It must be possible to modify an old knowledge base and to create a new knowledge base.

ii) The expert system must provide not only the computer's interpretation, but also facilities for verifying or excluding a particular interpretation.

iii) The expert system must be able to explain its logic.

5.1.3 Output of expert system.

The interpretation must be supplied to the user in a form that is readily understandable, and provision must be made for explanation of the physiological, and to some extent prognostic, implications of reaching a particular interpretation.

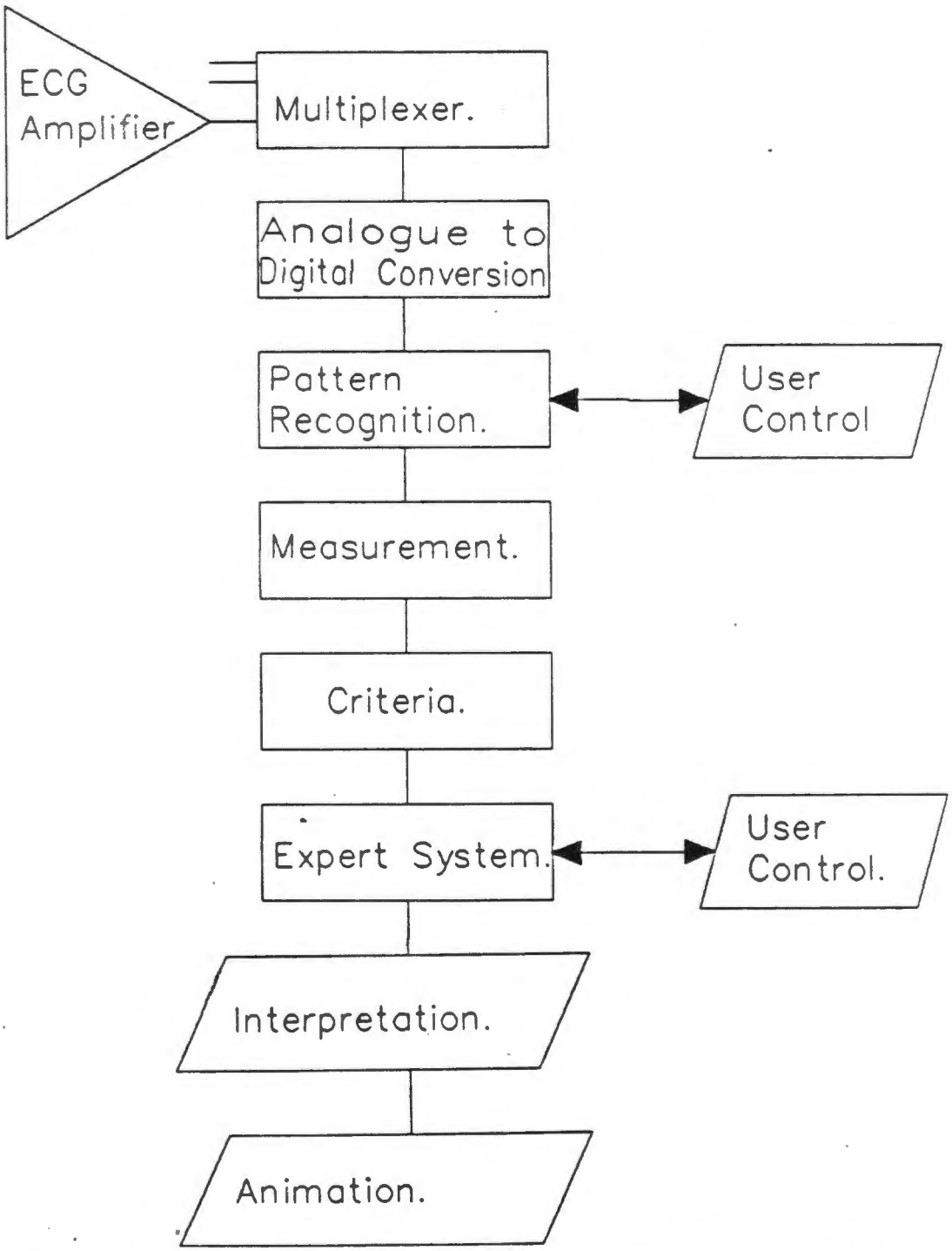

Figure 5.1. Diagrammatic representation of the tutoring interpretive electrocardiograph.

5.2 CLARIFICATION OF THE SPECIFICATIONS.

5.2.1. Input.

5.2.1.1 Frequency (bandwidth) response required.

The question of what frequencies, or frequency ranges, of the electrocardiograph signal contain useful information is not, for the clinician, a scientific question, but an historical question. The question can better be phrased: "For which frequencies of electrical signal recorded on the body surface does a sufficient body of data exist, so that reliable inference as to the existence of cardiac pathology can be drawn."

Most empirical studies in electrocardiology use commercially available ECG amplifiers. The American Heart Association specifies the frequency bandwidth for diagnostic ECG amplifiers to be 0.5 - 100 Hz (Sheffield et al 1978). The AHA recommendations have been criticised.

Some widely used ECG amplifiers have a non-linear phase response, and this distorts the signal (Taylor and Vincent 1983). More recent work on electrocardiology indicates that clinically useful information is contained at frequencies of up to 1500Hz (Shridhar and Stevens 1979). More work on high frequency electrocardiology may convince clinicians that routine examination of these components of the ECG signal will influence patient management.

New measurement standards for diagnostic ECG amplifiers are under consideration (Anonymous 1985) but until consensus is reached on new standards, the American Heart Association standards can be accepted.

5.2.1.2 Analog to Digital conversion.

Most systems for computer analysis of electrocardiographs store the digitized electrocardiograph signal at a fixed frequency (sampling rate), in a finite number of bits (precision).

i) Sampling rate.

The American Heart Association, as well as the American Food and Drug Administration, recommend a sampling rate of 500 Hz when converting analog ECG signals to digital format (Sheffield et al 1978). Barr and Spach (1977) recommends a sampling rate of 500 Hz for routine diagnostic ECGs, a rate of 834 Hz for neonates because of the higher pulse rate, and rates exceeding 6000 Hz for patients with ischaemic heart disease.

Shannon's sampling law requires the sampling rate to be at least twice the frequency of the highest frequency component required. Analog amplifiers made to American Heart Association standards output a signal in which the -3 dB down-point is at 100Hz. Shannon's law suggests that a sampling rate of 200 per second will capture these frequencies.

The most widely used computerised ECG systems use a sampling rate of 250 per second, and this rate can be considered to be the industry standard.

ii) Precision.

The American Heart Association recommends a resolution of 0.01 mV over a range of -5mV to +5mV of the input signal (Sheffield et al 1978). This represents a resolution of 1:1000, and this can be achieved if each digital sample is stored in 10 bits (2 to the 10th power = 1024). Many systems e.g. the IBM Bonner system and the American Veterans Administration AVA-4.0

system, use 12 bit precision, corresponding to a resolution of 1:4096 (Bonner et al 1978).

Berson et al (1977) studied the effect of changing the precision of the digitized ECG on the computer's performance in interpreting the ECG. He states that no significant deterioration was noted when precision was lowered to eight bits, but found seven bits unsatisfactory. However, Berson attributes the deterioration in the computer's performance with seven and six bit precision to the design of the computer's wave recognition algorithm.

Table 5.1 gives the approximate resolution possible with varying precision for a 10 mV peak to peak input signal

Precision	Resolution
10 bit	0.01mV or 1:1024
9 bit	0.02mV or 1:512
8 bit	0.04mV or 1:256
7 bit	0.08mV or 1:128
6 bit	0.16mv or 1:64

Table 5.1 : Resolution possible with varying precision.

Representation of ECGs with small P waves or subtle ST segment depression would be inadequate with less than 8 bit precision, as these features are manifest by deflections of 0.1 mV or less. Whether loss of these features will result in "significant" deterioration of the system as a whole depends on how many ECGs in the study have these features as crucial to their interpretation.

It seems sensible to be cautious, and accept 8 bit resolution

as the absolute minimum necessary in digitizing the ECG signal This is convenient as 8 bits is one byte. As the cost of both RAM and high speed processors plummets, is must also be questioned whether much is saved by using 8 bit rather than 12 bit or even 16 bit precision.

5.2.1.3 Entering ECGs from a paper record.

Many ECG records are stored as a tracing on paper, and it would be useful to be able to enter these into the system. Digital scanners, capable of resolutions of 300 dots per inch (dpi) are widely used and often configured to personal computers. This resolution is adequate to capture the ECG trace, and it is a relatively simple task to write a program which will recognise the line grid on the ECG paper, orientate the trace with respect to this grid, and to subtract the grid from the tracing. However, various problems with technical quality might arise. The moving pen used to draw the trace has a physical thickness, and discrepancies between systems which measure the top border of the line drawn, the centre of the line or the bottom of the line may be significant. The moving pen also has a slew rate, which will have the effect of flattening sharp peaks in the tracing. The amplifier used for the original tracing may also have characteristics which have distorted the signal

Before a system for reading ECGs from a paper trace is incorporated into the system, it will be necessary to construct the system and to establish empirically whether the distortion introduced is of an acceptable magnitude or not.

5.2.1.4 Storage.

The American Heart Association (ibid.) recommends that for

diagnostic electrocardiology a 12-lead recording, with at least 2.5 seconds per lead, is required. The frontal leads have to be recorded, but four of the remaining six leads carry redundant information, and can be generated algebraically.

Thus storage of a 12 lead diagnostic ECG requires the storage of 2.5 second of data from each of eight leads. This implies a total of 20 seconds of signal If this is sampled at 250Hz with a 12 bit precision, then

20 x 250 x 12 = 105 000 bits, (13 125 bytes)

are required for each ECG. This indicates that over a thousand such ECGs can be stored on a single 20Mb hard drive. Use of a suitable data compression algorithm could increase this figure considerably. The IBM/Bonner System (Bonner et al 1978) manages to store the electrocardiograph data, as well as associated patient data, in 1464 bytes. Other systems use between 1250 bytes and 78kb (Pryor et al 1980). Use of data compression algorithms can be expensive in terms of computer time. Data compression algorithms that achieve very high compression ratios (i.e. can store 13 kb of data in 1250 bytes) do lose some information, and the reconstructed signal is occasionally not of acceptable standard.

5.2.1.5 Pattern recognition.

Assuming that basic signal processing such as filtering has been done on the analog signal before digitizing, the first task of the computer is to identify the P wave, QRS complex and T wave, and U wave, if present. Various pattern recognition algorithms have been devised to do this. The process would be better described as "feature detection", but the term "pattern recognition" seems to have stuck.

Features which pattern recognition algorithms use are amplitude,

slope or change of slope above a critical value, and deviation from the baseline.

Pattern recognition algorithms have been discussed in detail. To summarise that discussion:

i) Failure of Pattern recognition algorithms to correctly identify the waveforms are a major cause of incorrect interpretation by computer.

ii) This failure is attributed in part to the failure to compensate for excessive noise and artifacts in the signal

It is necessary for teaching purposes for the student to be able to identify the wave components visually. To allow the user to inspect the trace and identify wave shapes requires some means of visual display of the ECG trace.

5.2.1.6 Visual display.

The previous discussion on digitizing the ECG indicates that a sampling rate of 250Hz at 8 bit precision is adequate to capture the information necessary for clinical electrocardiology.

The display of this information would therefore require 250 pixels horizontally per second, (or 625 for the 2.5 second strip), and 256 pixels (2 to the power 8) vertically.

Thus each lead (of 2.5 seconds duration) of the 12 lead ECG would require a screen resolution of 625 x 256 pixels. To display all twelve leads simultaneously, using the common arrangement of three rows of four leads, would require

$$(4 \times 625) \times (3 \times 256) = 2500 \times 768 \text{ pixels.}$$

This level of resolution is not often obtainable. Table 5.2 gives the resolution possible with common graphics cards used in IBM PC

compatible microcomputers.

Graphics Card	Highest resolution supported
IBM CGA	640 x 200 pixels
AT & T	640 x 400 pixels
Hercules	720 x 350 pixels

Table 5.2. Resolution supported by different graphic cards.

Thus two modes of display are required. A crude overview of the entire ECG could be obtained by displaying the 12 lead ECG at resolutions below the required 2500 x 768 pixels. A second mode, displaying only one of the twelve leads, would be necessary for correct identification of the components of the waveform.

5.2.1.7 Waveform measurements.

Once the waveform components have been identified, measurements can be taken. Hewlett-Packard (1979) define 420 such measurements. (Three sets of forty rhythm criteria, plus thirty-four morphology criteria for each of the twelve leads.) Unless the superiority of a novel approach can be adequately demonstrated, this approach should be followed.

These measurements must now become the input to the expert system module. The expert system module's criteria do not necessarily correspond neatly with the measurements taken, i.e. the criterion "raised S-T segment" comments on the area under the S-T segment, but also on the concavity of the S-T wave shape. If the expert system is provided with a list of measurements, and the task of converting measurements to criteria is a function of the knowledge base and expert system, the knowledge base will become

more complex and more difficult to modify.

If this conversion is done in an intermediate linking module, then all knowledge bases are obliged to use the same criteria definitions. One solution is to split the knowledge base into two parts. The low level part would convert measurements to criteria, and a higher, more easily modifiable part would reach interpretations from the criteria provided.

All methods require the measurement data to be stored somewhere, and the expert system has to know how this is stored in order to access the information. This places further constraints on the manner in which the knowledge base can be created.

5.2.2 Expert system module.

An expert system consists of an inference engine (expert system shell) and a knowledge base. In the case of interpretation of electrocardiographs, there are of the order of one hundred and fifty possible comments or interpretations that can be made about an individual tracing. Most, but not all of these interpretations are mutually exclusive.

The knowledge base consists of a set of all comments that can be made about the ECG (questions), a set of all interpretations that can be applied to the ECG (decisions), and the logical connections between the comments and the interpretations (rules).

The inference engine is the mechanism for linking the set of comments on an individual electrocardiograph to the correct interpretation or interpretations.

5.2.2.1 Discussion on types of inference engines.

i) Simultaneous Inference engines

The simplest type of inference engine is the simultaneous inference engine. This system represents the knowledge base as a matrix of X number of comments and Y possible interpretations (Table 5.3).

	Interpretation 1	.....	Interpretation Y
Comment 1			
Comment 2			
Comment 3			
.			
.			
.			
.			
Comment X			

Table 5.3 : Knowledge base represented as a matrix.

Each cell in the matrix contains a value 0 or 1 depending on whether that comment is applicable to the interpretation. The sample ECG is represented as a (X,1) matrix. The generation of this matrix requires input on the presence or absence of all comments 1 to X.

Multiplying the knowledge base matrix by the sample ECG matrix produces the answer matrix. The highest value in the answer matrix corresponds to the correct interpretation.

This type of inference engine has the disadvantage of requiring all the values for the comments 1 to X as input, even when many of the comments may be inappropriate to the

sample ECG in question. For this reason alternate inference engines have been developed, which require only relevant information as input.

ii) Rule based inference engines.

Rule based expert systems represent the knowledge base in a tree structure. The points in the tree are called nodes. In the example used above each node would be a comment on an individual electrocardiograph. Each node has two outcomes:- the comment can be either true or false for the sample electrocardiograph.

The inference engine works through the tree, and each node is only set, i.e. presence or absence of the comment noted, when the inference engine reaches that node. The correct interpretation is reached in the shortest (or almost shortest) route, and irrelevant information is not required.

There are two basic strategies for chaining through the tree. Forward chaining begins at the node which will have the greatest effect in cutting down the number of possible interpretations. It will then move on to the node which offers the next greatest effect. While this represents the fastest algorithm for chaining through the tree, it presents the user with seemingly unconnected requests.

Backward chaining begins with the interpretation, and then attempts to verify that interpretation by setting the nodes which lead to the interpretation. If a node is set which is inconsistent with reaching the interpretation, no further attempts are made to set the nodes concerned with that interpretation. Backward chaining does have drawbacks. If the

user is uncertain as to what interpretations are likely, the user would be unable to select an interpretation for the expert system to attempt to evaluate.

This drawback can be sidestepped by providing a default mode where the expert system will begin with the first interpretation on the list. If this proves incorrect, it will attempt to validate the next interpretation. If a correct interpretation is found, the user will be asked if further possible correct interpretations should be sought. In practice this will appear to function like the forward chaining algorithm, but will be slower if less erratic.

iii) Recommendations.

Of the two systems described, the first is by far the simpler, and is therefore much easier to code. For teaching purposes, the rule based system, offers many features which the make it the obvious choice. Apart from allowing the knowledge base to be separated from the inference engine, rule based systems demonstrate the relevance of particular questions simply by not asking unnecessary questions. Additional explanation facilities to explain why a particular question is being asked are also provided. The student can is also urged to consider the likely interpretation of the ECG before any interpretation begins in the backward chaining mode. Finally, the trace of the line of questioning followed by the expert system can be supplied to explain how the system arrived at a particular interpretation.

5.2.2.2 The knowledge base.

The creation of the knowledge base (knowledge engineering) is

perhaps the most difficult part of the expert system. Any expert system can be only as good as the knowledge base it utilizes, and for non-trivial knowledge domains it is not possible to write a knowledge base which is a definitive account of the domain. Any knowledge base is only its author's subjective account of the field in question.

This last point must be emphasized. Because of the popular mystique about computers, the user is tempted to view an expert system generated interpretation as a function of electronic wizardry. The user's receptiveness to such an expert system's advice will therefore depend on the user's perception of computers in general This is both unhealthy and false.

That the expert system's opinion is actually the opinion of the author of the knowledge base used can be emphasized by forcing the user to select a knowledge base, rather than providing a default or turnkey option. The knowledge base should also be named in a manner which describes authorship, i.e. "ABC General Hospital Electrocardiograph Knowledge Base," rather than "Comprehensive Electrocardiograph Knowledge Base".

The availability of more than one knowledge base is also desirable. Thus a user may conclude "The General Hospital Knowledge Base says Y, while the XYZ Medical School concludes X, and I feel X more likely." This is a much more favourable reaction than that of a user who feels "The computer says Y, which seems wrong", or even "The computer says X and Y, and can't make up its mind."

At present many knowledge bases for electrocardiograph interpretation exist in the public domain. With further

development, more are likely to appear, and existing knowledge bases will be modified for local conditions. Where competing knowledge bases exist it is desirable from a medical point of view to have access to both. The copyright status of a knowledge base in a proprietary system is at present unclear.

5.2.3 Tutoring and explanatory functions.

5.2.3.1 Explanatory functions available in the expert system shell.

Expert systems constructed on the MYCIN model (i.e. the rule based system described above) provide two types of explanatory functions. The first explains the logical construct used to reach any particular conclusion - i.e. displays the rule or rules used to reach a conclusion. This explanation may take the form "Conclusion X because: Criterion A AND NOT(Criterion B OR NOT Criterion D) ". The second explanatory function consists of text attached to each question. A query response to a question displays this text. A second query response displays the text associated with the previous question as well as the current question.

5.2.3.2 General requirements for a tutoring system.

While explanatory facilities exist in the inference engine, these facilities explain the functioning of the expert system. A tutoring system requires more than this.

The student must also be taught how to read an ECG trace, and to recognise how abnormalities are manifest in the trace. The understanding of a particular interpretation of an ECG also requires an understanding of the underlying mechanisms of

arrhythmogenesis, morphological or biochemical abnormalities, and an appreciation of how these electropathologies manifest on the ECG trace.

Animated computer graphics can illustrate the movement of the heart, and visualisation of the events which occur during an arrhythmia provide considerable insight into the pathology. Animation can also be used to illustrate the spatial relationship between cardiac morphology and the position of the electrodes on the body surface, and how this relationship is altered by morphological abnormalities. Biochemical disturbances are more difficult to illustrate, but the display of electrolyte movement on a cellular level will be helpful.

Functions to support these requirements must be provided by the Tutor.

5.2.3.3 Tutoring functions.

A critiqueing model is proposed as the best way to teach the student to read the ECG trace. The student is required to identify the waves without access to the computer's opinion. The computer then compares the student's wave identification with the results of its own identification. If the two agree, the system proceeds to the next stage. If agreement is not present, the student is informed of the discrepancies. Facilities for the system to proceed solely on the basis of the student's identification should also be provided, for two reasons. The student may feel his own identification to be superior, and should have the opportunity to explore the implications of this decision. Secondly, excessive noise and artifacts are often obvious to a human observer, but are quite tricky to detect by computer (Rautaharju and Okajima 1980). In this case the student

may be correct in rejecting the tracing as being of inferior quality.

Tuinstra et al (1982) compared computer identification of the wave shapes with visual identification, and found the visually identified wave shapes to be substantially shorter than the computer identification. This may be a result of the pattern recognition algorithm used in the study, or may have been caused by the fact that the human observer identifies deviations from the baseline from the point at which they become pronounced, while the computer uses a much finer measurement. If these findings are correct and due to error in visual coding, the module which compares the student and computer identification of the wave shapes will have to tolerate some degree of error in this direction.

Facilities must exist to switch from a question asked by the expert system to the display of the ECG rather than deriving answers directly from the measurement matrix of the ECG in question, in order to allow the user to reach his own answer to the query. The system must then correlate user supplied answers with answers obtained from the system's measurements on the ECG.

Animated graphic sequences are to be available in the system to illustrate every interpretation which may be reached by the system. Display of the animation must be dependent on first arriving at a correct interpretation. Thus the student is obliged to work through the steps of identifying the wave components of the trace, and answer questions about the abnormal findings, before being rewarded with the explanation of the electropathology present.

CHAPTER 6

A DESCRIPTION OF THE SYSTEM IMPLEMENTED

A prototype system has been created which implements some of the specifications outlined in Chapter 5. The system consists of two modules, the Input module and the Expert System module. Apart from difficulties in interfacing the two modules, the system is operational. A third module, which is capable of creating animated graphics, has also been written, and is described.

Custom hardware and software was developed for the input module. An amplifier is used to take an electrocardiograph reading from a patient. The amplifier is interfaced to an analog-to-digital converter, which converts the reading to digital format, and stores the data on floppy disk formatted under MSDOS. A program, written in Turbo Pascal 3.0 under MSDOS, accesses this data from disk and displays the electrocardiograph trace on the computer screen. Rudimentary waveform identification and measurement functions are provided.

An expert system shell is implemented in Turbo Pascal 3.0 under MSDOS. Two knowledge bases concerned with clinical electrocardiology are provided for the shell. The shell has facilities for displaying an interpretation (decision) in the form of an animated graphic. An animated graphic series of twenty-four pictures, representing a particular interpretation of the electrocardiograph, is provided.

A program to create animated graphic displays compatible with those required by the expert system shell is implemented in Turbo Pascal 3.0.

6.1 THE INPUT MODULE

6.1.1. Recording the analog signal and storing the digitized signal to disk.

Hardware developed at the Department of Biomedical Engineering, U.C.T. was used. This consists of:-

6.1.1.1 A six channel ECG amplifier.

Commercial electrocardiographs (Hewlett-Packard 1984a) use a three channel amplifier with automatic switching between leads with negligible information lost in the switching. The twelve channels are obtained as four sequential sets of three leads each. No such facility was available on the amplifier used in this project. For the purpose of further developmental work, this was not a significant drawback. Further development proceeded by assuming that the incomming signal would consist of two readings from leads in the frontal plane, and readings from the six anterior (V) leads. This configuration was simulated by taking readings from the six channels available, two of which were from leads in the frontal plane. Four of the remaining six frontal plane leads could be derived from the data thus obtained. The remaining two leads would not be recorded. Data for these leads would be "simulated" by duplicating and re-naming data already recorded.

6.1.1.2 An A-D conversion card.

An A-D card (developed by M. Popp of the Department of Biomedical Engineering, U.C.T.) installed in an IBM PC was used to interface with the ECG amplifier. Six leads were digitized simultaneously at 250 Hz per channel with 12 bit precision, for 2.5 seconds. The digitized data was stored on floppy disk, with the data from each

lead in a separate file. Two files were duplicated to simulate the data for the two anterior chest leads which were not recorded, giving a total of eight files. As the amplifier used to record the ECG was not wholly suitable for the task, the system was not calibrated.

6.1.2 Displaying the ECG.

6.1.2.1 Input of data

The data from the analog-to-digital converter are stored to disk in eight files, representing eight of the twelve leads required.

It is desirable for rhythm analysis to have a recording of more than 2.5 seconds. The system as described provides for only 2.5 seconds to be recorded for any particular lead. As the leads are conventionally displayed simultaneously in a 3 x 4 arrangement (see figure 6.1) it can be seen that leads Std. II, aVl, V2 and V5 are on the same horizontal plane. If the time taken to switch between leads in this set is negligible, a continuous 10 second reading can be represented in these four leads. Thus a three channel amplifier with rapid switching is adequate to record all the leads required, and to have data suitable for rhythm analysis.

The display module displays the electrocardiograph as four sequential sets of three leads each. Although hardware used for this project did not offer facilities for rapid switching between leads, it was possible to generate data which could be used to test the functioning of the display module.

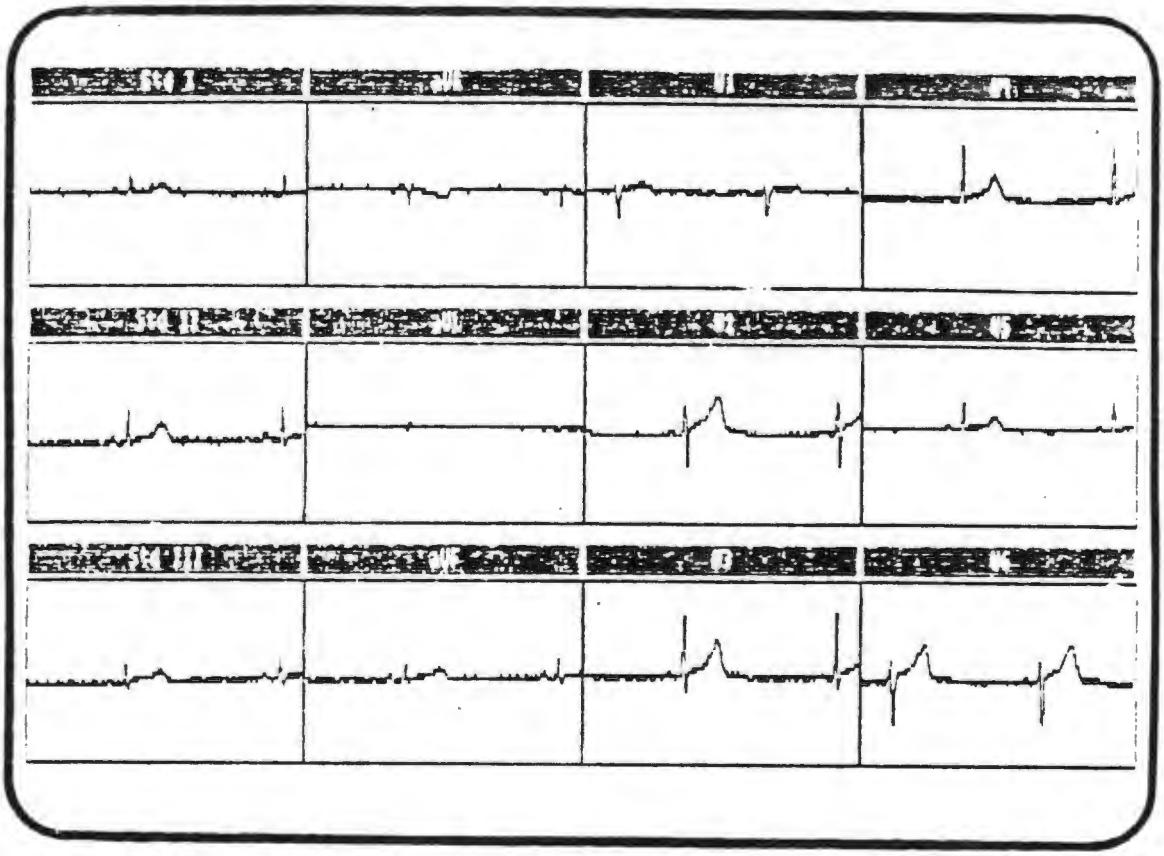

Figure 6.1 Screen dump of the display of the twelve lead electrocardiograph.

note: The trace displayed represents data chosen to test the functioning of the software only, and does not represent a continuous ECG.

a) Conversion of data.

As the dynamic range of the amplifier output is 10 V and this is converted to digital form with 12 bit precision, it follows that

10 V (output range) = 4096 units (2 to the power 12)

However, as the 10 V range extends from -5 to +5 volts it follows that

0 V (output) = 2048 units

The amplifier has a gain of 1 000, therefore

-5 mV (input) = -5 V (output) = 0 units.
0 mV (input) = 0 V (output) = 2048 units.
+5 mV (input) = 5 V (output) = 4096 units.

The following formula converts the value of the digitized sample value into millivolts:

Amplitude (mV) = (Digitized wave-2048)/0.4096

This process of reading the data from disk and converting the values to millivolts takes approximately 5 seconds per lead.

b) Calculating the remaining four leads.

Six leads of the twelve lead ECG represent readings from the frontal plane. The three unipolar leads, aVr, aVl and aVf, represent potential difference at the right arm, left arm and left leg respectively, measured against a neutral body earth provided by the sum of all three unipolar leads. These three points approximate a triangle in the same plane as the heart (Einthoven's triangle, displayed in figure 6.2). As the sum of the potential differences at all three points of a triangle containing an electrical charge must equal zero, the third lead can always be calculated, given the other two.

The three standard leads, Std. I, Std. II and Std. III, represent the difference between two of the three points. By convention,

Std. I = aVl - aVr

Std. II = aVf - aVr

Std. III = aVf - aVl

The display module requires values for leads Std. II and Std. III. The leads Std. I, aVr, aVl and aVf are then calculated. The data from all 12 leads now exist in the computer's memory, and represent values in millivolts.

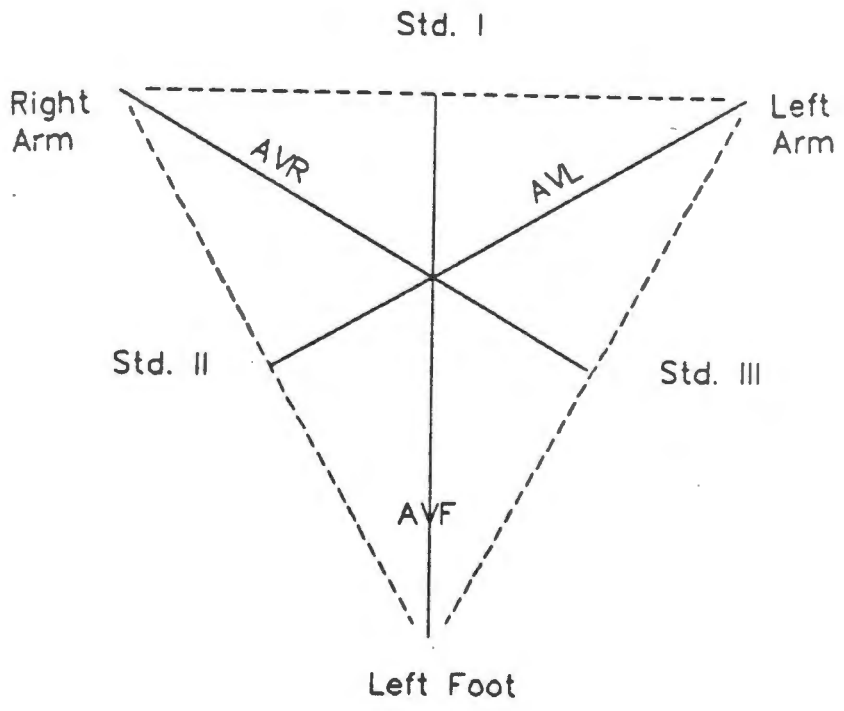

Figure 6.2 : Einthoven's triangle, showing the orientation of the six frontal leads.

6.1.2.2 Displaying the ECG Waveform

The ECG data in RAM is displayed on the screen. This is done by:

a) Drawing a template on the screen.

A window is set up, comprising one twelfth of the screen, i.e. 125 x 66 pixels. The window has a border and a heading. This is repeated twelve times, with the appropriate heading for each window. The screen thus displays twelve non-overlappling windows, each of which represents one of the leads of the twelve lead ECG.

b) Drawing the waveform.

Three panels are set up, each comprising a row of four of the 125 x 66 windows drawn for the template. Each of the three panels are given co-ordinate axes in "seconds" horizontally and "millivolts" vertically. Correct scaling thus takes place automatically. The waveform is then plotted, starting from the left. The first point is plotted in the top panel, then the "active" panel is switched to the second panel, and the first point plotted there, then to the third panel. The process is repeated with the next point to be plotted. The switching of "active" panels happens very fast and gives the impression of three waves being plotted simultaneously.

It takes 30 seconds to plot the entire ECG. The slow speed is partly due to hardware limitations, and partly because the range of values in both the horizontal and vertical axes greatly exceeds the number of pixels available for their representation. Thus two adjacent points on the waveform might be represented by the same pixel - the pixel will therefore be

plotted twice.

A screen dump of the plotted waveform is shown in Figure 6.1.

6.1.2.3 Wave identification functions.

Two main tasks should be performed:

a) A review of the performance of the pattern recognition module. The pattern recognition module identifies the P, QRS and T waves, and the U wave if present. The identification can be done manually by labelling points on the ECG trace "P wave start", "P wave end", "QRS complex start " etc., but this will result in a confusing array of labels. A better method will be to highlight the wave by colour-coding or utilising a dashed or a thicker line, and label the wave as a whole. This approach will produce a simpler display.

For teaching purposes it is necessary to allow the module to function in a "critiqueing" mode, i.e. the student is required to identify the waveforms first, and only if this is not done correctly are the results from the computer's identification displayed. Another desirable function is having the choice of displaying the results of two or more pattern recognition algorithms. Many pattern recognition modules have been devised, and none perform well on unusual ECGs. The facility to toggle between different displays will allow the user to chose the best algorithm.

As work on pattern recognition algorithms would greatly extend the size of the present thesis, this section has not been implemented.

b) User controlled wave identification.

This facility offers the user the option of modifying the wave identification of the pattern recognition module, or identifying all the waves separately.

Functions which simulate the use of draughtsman's dividers have been implemented. A "caliper", representing a point of the dividers, appears simultaneously as a vertical line in the three horizontal panels of the displayed ECG. The "caliper" can be moved horizontally to any point on the ECG. A second "caliper" can be selected. It represents the second point of the dividers, and can be moved while the first caliper remains stationary. The distance between the two calipers is displayed at the bottom of the screen. Once the second caliper is at the required point, an option can be selected to move both calipers simultaneously, keeping the distance between them constant. This facility is useful in determining rhythm abnormalities, as one R-R (or P-R) interval can be measured with a set of calipers, and then both calipers moved simultaneously to the next R-R (or P-R) interval to establish whether or not the two R-R (or P-R) intervals differ. Two sets of calipers are available.

A "zoom mode" allows any lead to be enlarged to enable the calipers to be positioned with greater accuracy. A screen dump of the zoom feature is shown in Figure 6.3.

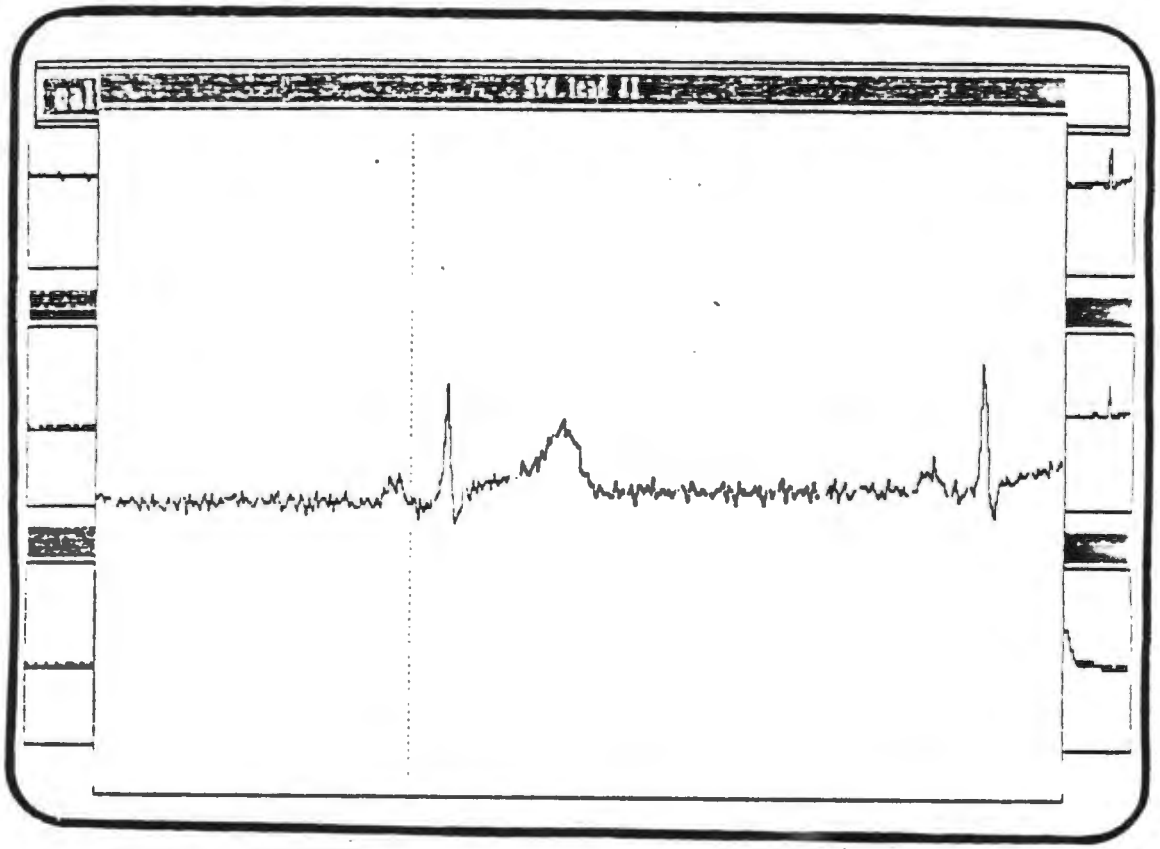

Figure 6.3 A screen dump of the display of a single lead in zoom mode.

Facilities for marking and recording the onset and termination of waves have not been implemented.

The module is controlled by a pop-down menu system which provides a state of the art user interface. A screen dump of the pop-up menu facility is shown in Figure 6.4.

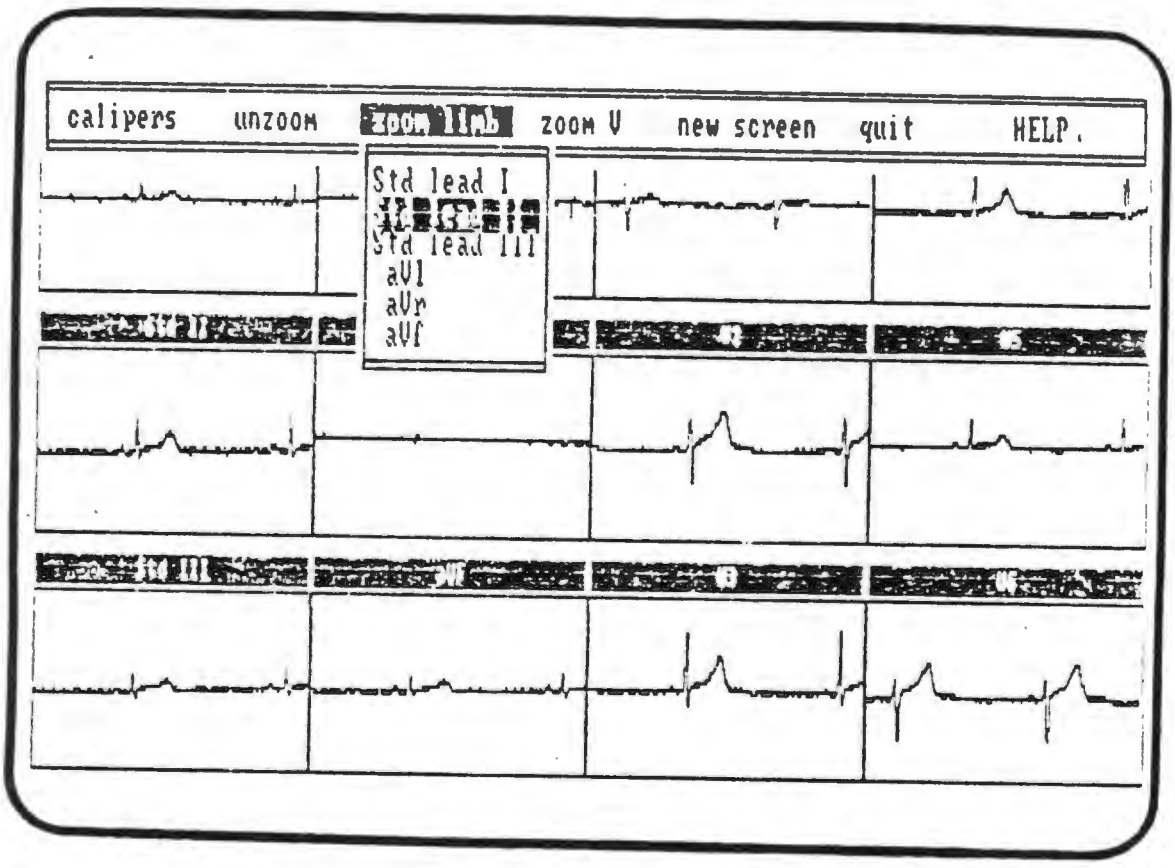

Figure 6.4. A screen dump of the pop-down menu system.

6.1.3 Flowchart.

The flowchart for this program is contained in appendix I.

6.2 THE EXPERT SYSTEM MODULE

The software module comprises the following:

i. Knowledge bases (Two currently on system).
ii. Expert system shell
iii. Animation sequences.

6.2.1 The knowledge bases

Two knowledge bases have been written for the system.
The first is a simple example to clarify and validate the system. It is designed to determine whether or not an electrocardiograph indicates the presence of Wolff-Parkinson-White Syndrome. It consists of four rules, five questions and four decisions, of which one is an intermediate decision. The knowledge base is derived from "The Wolff-Parkinson-White Syndrome", Schamroth 1980.

The second example is more substantial. It consists of seventy-five questions, fifty-five rules and sixty-one decisions, of which thirty-six are intermediate decisions. It is designed to determine whether or not the electrocardiograph indicates the presence of various types of myocardial infarct. It is based on the Marquette Diagnostic Criteria.

6.2.1.1 The knowledge base language.

The knowledge base consists of three classes of statements, viz. decisions, questions and rules.

Decisions are written in the form

"Decision <number> : <quote>."

The word "Decision" is a reserved word in the knowledge base language. The shell will accept any abbreviation beginning with the letter "D". All reserved words can be typed using either lower or upper case letters.

The <number> represents a decimal integer from one to two hundred. The upper level (two hundred) is an artificial limitation imposed by the shell, and can be increased if the host machine has sufficient RAM.

The <quote> is a line of free text in quotation marks. In this example each quote represents an interpretation of the ECG.

Thus a decision may take the form:

D1: "Normal electrocardiograph."

Questions are written in the form:

Question <number>: <quote>
Answer 1 <quote>
2 <quote>
.
.
9 <quote>

Why <quote>

The words "question","answer" and "why" are reserved words, and

may be abbreviated to their initial letters. Each question may have between one and nine possible answers. The shell does not allow the user to choose more than one correct answer to each question. Thus a question which could have as correct both answer 1 and answer 2, should have, say, answer 3 as being "Answers 1 and 2 both correct".

The "why" line offers the option of including an explanation of why this particular question is being asked.

All quotes consist of a line of free text, in quotation marks, which poses limitations on the way questions can be phrased. It is often difficult to pose the question in a single line of text, and more difficult to adequately explain the reason for asking the question in a single line. This restriction has been imposed to simplify the coding, but can be altered to allow a variable number of lines of free text.

The rules are the 'cement' which connects the questions to the decisions. Rules consist of a decision on the right, connected to a logical construct of question-answer nodes on the left by an IF-THEN statement.

Question-answer nodes are written in the form:

    Question<number>answer<number>,

or abbreviated to the form:

    q<number>a<number>.

The logical constructs NOT, & (logical and), OR (inclusive or) are supported, and nested brackets can be used.

The sample rule:

RULE 4 :IF q3a1 AND q5a9 THEN D10.

states that decision number 10 is correct if the answer to question 3 is number 1, and that to 5 is 9.

The logic can become complicated with the use of nested brackets.

The example

RULE 154 :IF NOT((q3a1 AND q5a9) OR q120a1) THEN D2.

can be simplified by noting that RULE 4 states that decision number 10 is true if q3a1 and q5a9 are true. Thus rule 154 can be written as

RULE 154 :IF NOT(D10 OR q120a1) THEN D2.

Decisions which may have no significance to the knowledge domain (i.e. they do not represent a possible interpretation of the ECG), but are useful in simplifying the logic, are intermediate decisions. In the above example, if Decision 10 (declared in Rule 4) was declared solely to simplify the writing of Rule 154, Decision 10 would be declared as an intermediate decision. Intermediate decisions are declared in the same manner as other decisions, but are not associated with a text quote.

6.2.2 The expert system shell

The expert system shell is implemented in Turbo Pascal running under MSDOS. In order to have an expert system shell suited to this purpose, original features were added to an existing expert system. The shell is based on a public domain system written for the UCSD p-system. The public domain system implements the

features pioneered by the EMYCIN expert system shell, but does not support fuzzy logic. EMYCIN is a knowledge-domain independent shell derived from the MYCIN project. The original shell was converted from the UCSD p-system to MSDOS, and further modified to comply with the Turbo Pascal standard. Original features added include revised screens, the ability to display the knowledge bases available on disk and to select the knowledge base from a menu, the ability to go backwards, i.e. to return to the question just answered in order to change the answer given, and the provision of a "Hint" facility.

An expert system shell has some features of a high level language compiler, and some features of a data base management system.

The expert system shell accepts as input a knowledge base. The knowledge base must be written to conform to the syntactical requirements of the expert system shell, i.e. it must be written in a language the shell understands. The knowledge base is "compiled", i.e. converted into a form, resident in the computer's memory, that the computer can manipulate.

Once the knowledge base is compiled into the computer's memory, the shell provides methods for moving through the knowledge base, a feature analogous to those of a data base management system. I discuss the system by first introducing the data structures used to store the "compiled" knowledge base. Strategies used to detect errors in the knowledge base, and methods used to move through the knowledge base are then discussed. This discussion gives an overview of the module structure. The flowchart can be found in Appendix I, detailed documentation of the actual coding is contained in Appendix II and the full program listing in Appendix III.

6.2.2.1 Data structures used.

Arrays are used to store decisions and questions declared. In order to conserve space in the data segment, which Turbo Pascal limits to 64kb, the actual text associated with each question or decision is stored on the heap. The size of the heap is limited only by the amount of memory available on the host computer. The arrays contain pointers which indicate the address of a particular question or decision on the heap.

Each rule is stored as a binary tree. The root node of the tree represents a decision. The nodes of the tree represent question-answer nodes or decision nodes. Decision nodes within a tree are treated as intermediate nodes. Whether they are declared as intermediate decisions or as final decisions depends on whether the decision is meaningful to the knowledge domain, or whether the decision is simply a useful technique for simplifying the writing of the rules. All decision nodes in the tree represent the root nodes of another tree. Figure 6.5 contains a diagramatic display of two decision trees.

6.2.2.2 Error checking strategies.

Three types of errors are checked.

a. Syntax errors.

These usually arise from typing errors or failure to close quotation marks. Where these errors are detected they are noted as NON-FATAL ERRORS, but ignored. However, the consequences can be serious. If the text associated with a question does not commence with a quotation mark, the text will be ignored. When that question is asked, the user will be confronted with a list of possible answers, but no phrase

indicating what the question is. The term "NON-FATAL" does not imply that the user can proceed. It is an aid to developers of knowledge bases, and indicates the presence of an error which must be corrected. A knowledge base should not be released to the user until all errors have been corrected. The system will not crash in the presence of a "NON-FATAL" error, but will continue to search for other errors.

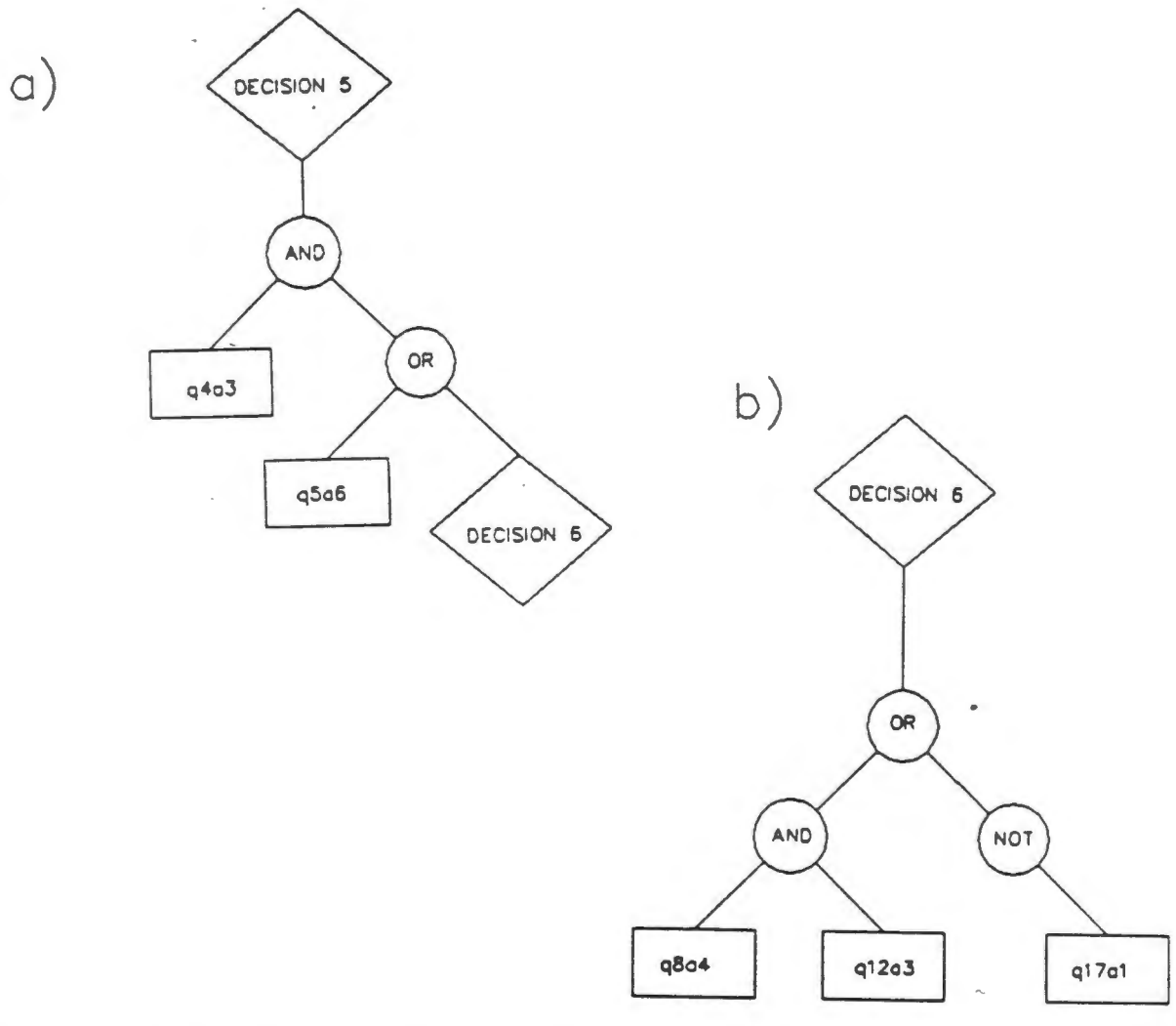

Figure 6.5 Diagram of two decision trees.

note:- Decision tree (a) calls decision tree (b). This may have been done because decision 6 is a meaningful point in the knowledge base, or decision 6 may be an intermediate decision, introduced to simplify the rule for decision 5 by splitting the tree into two parts.

b. Declarative errors.

Once the knowledge base is in memory it is audited. It is done by scanning through all the binary trees of all the rules, and noting which questions and decisions are used. If decisions or questions exist in the question or decision arrays, but are not used in any of the rules, or if questions or decisions are used in the rules but do not exist in the arrays, an error results.

c. Logical errors.

As decisions can be defined in terms of other decisions, it is possible to write the following rules:

```
RULE 1:IF D1 THEN D2.
RULE 2:IF D2 THEN D1.
```

or the rule

```
RULE 1:IF q1a1 AND (NOT q1a1) THEN D1.
```

Such circular logic will be detected by the audit procedure and will result in an error.

Multiple definitions are also not allowed. The example

```
RULE 1:IF D1 THEN D2.
RULE 2:IF NOT D3 THEN D2.
```

may make sense in a particular context. The example

```
RULE 1:IF D1 THEN D2.
RULE 2:IF NOT D1 THEN D2.
```

is clearly nonsense. The shell is unable to distinguish between the examples, and will detect an error in both. The correct way to code the first example would be

```
RULE 1:IF D1 OR (NOT D3) THEN D2.
```

6.2.2.3 Methods for chaining through the knowledge base.
Each rule is a binary tree, with a decision node as the root node, and every decision node is the root node of only one tree.

The simplest method of moving through the knowledge base is therefore to choose a decision node, and to work through the binary tree associated with that decision. This method is termed backward chaining.

Backward chaining has the disadvantage of requiring the user to decide in advance what decision to choose. If that decision proves incorrect then a second decision must be selected. The shell has the facility to remember which questions were asked while considering the first decision, and not to ask again questions already answered, but if three or four wrong decisions are chosen the process can get tiresome.

A second, pseudo-forward chaining method is offered. In this method the system chooses the first decision from the list. As soon as this decision proves incorrect, the second decision is chosen. If questions already answered by the user invalidate this decision, it is immediately rejected. If not, the system attempts to validate the decision by asking the user the relevant questions. If a correct decision is found, the user has the option of stopping at that point or continuing to search for further possibly correct decisions. Pseudo-forward chaining is, in essence, automated backward chaining through the entire knowledge base.

True forward chaining involves searching for the questions which would have the greatest effect in cutting down the number of possible correct decisions, and then asking those questions

first. This option is not offered by the shell.

6.2.2.4 Linking the shell to the ECG display module.

The function of the ECG display module is to take measurements of the ECG trace, and then make these measurements available to the expert system module. This linkage requires a special module for converting measurements in milliseconds or millivolts into a form that the system can use for answering the questions posed by the knowledge base. Again problems arise as to how this can be done without making the knowledge bases very complicated to both write and modify. These problems require further work to reach a solution, and the implementation of the module is outside the scope of this thesis.

The expert system shell, however, has been provided with a "handle" , or point where such a module can attach.

Whenever the user is required to answer a question, he can choose the "hint" option. As currently implemented, this option interrogates the rule tree to determine which answer is compatible with the decision (interpretation) currently pursued.

The option is intended, in a final version, to interrogate a file containing information about an actual ECG, created by the "Display ECG" module. An "automatic" mode is envisioned, whereby the answers obtained from this file are provided to the expert system without requiring any input from the user.

6.2.3 Using the system.

When the program is loaded an introduction screen appears (figure 6.6). After this screen, a menu screen appears (figure 6.7). The user is shown a list of knowledge bases available on the logged

drive, and is asked to select one of these.

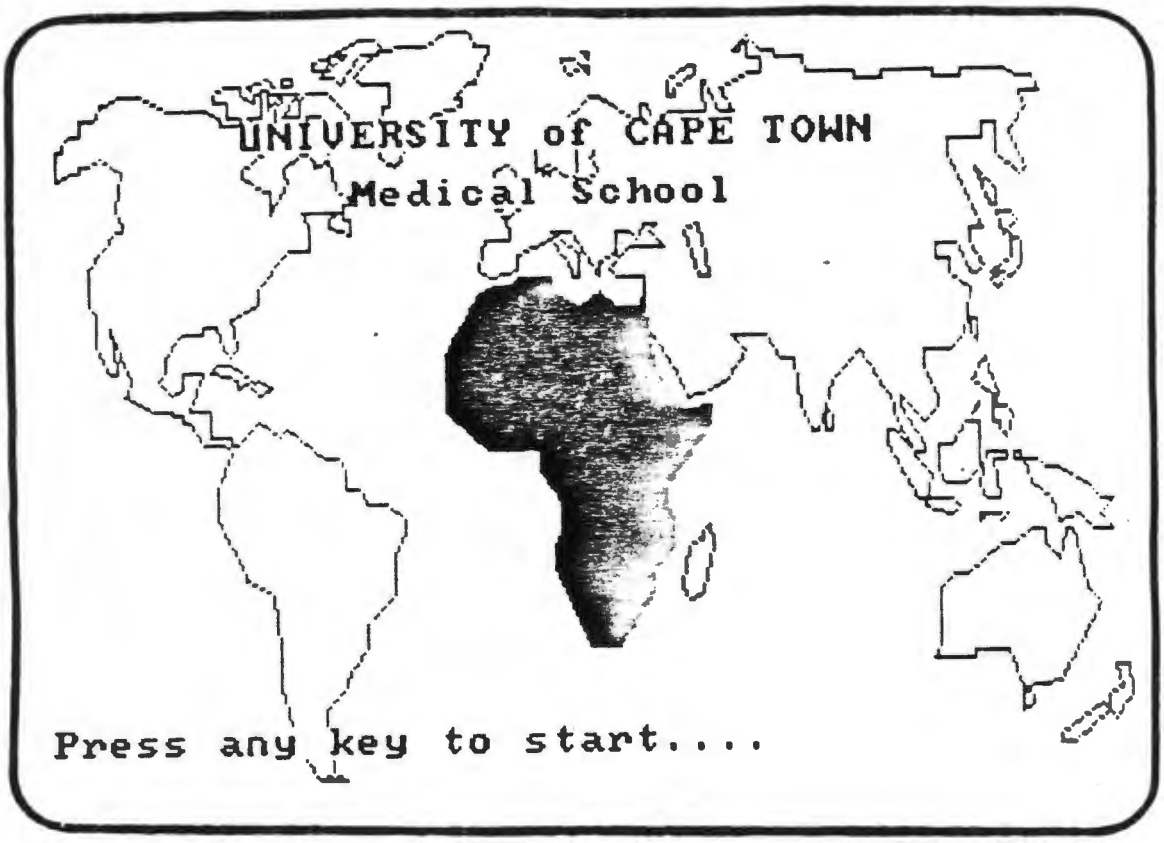

Figure 6.6 The opening screen.

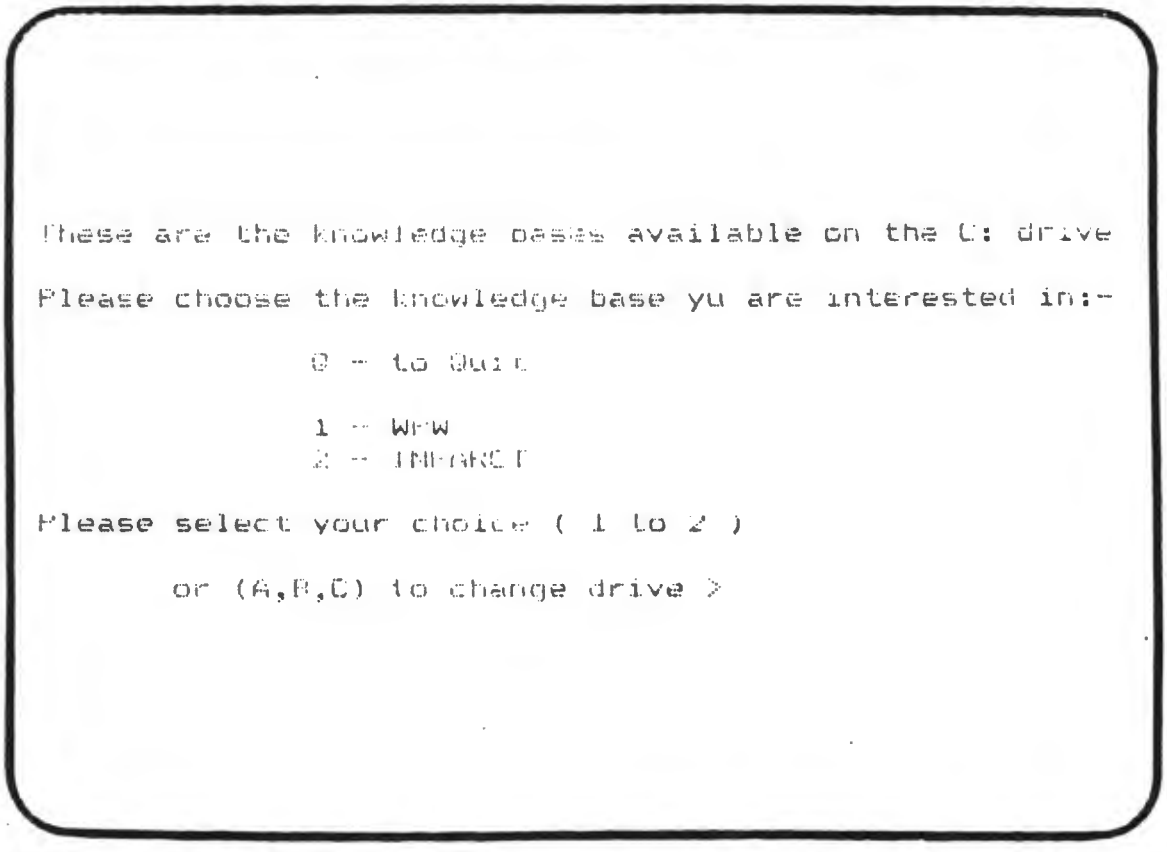

Figure 6.7 : Menu of knowledge bases available.

note :- Two knowledge bases are available in this example.

Once the selection is made the knowledge base is loaded. The user then has the option (figure 6.8) of choosing an interpretation from a list (i.e. backward chaining) or of getting set-by-step guidance (pseudo-forward chaining).

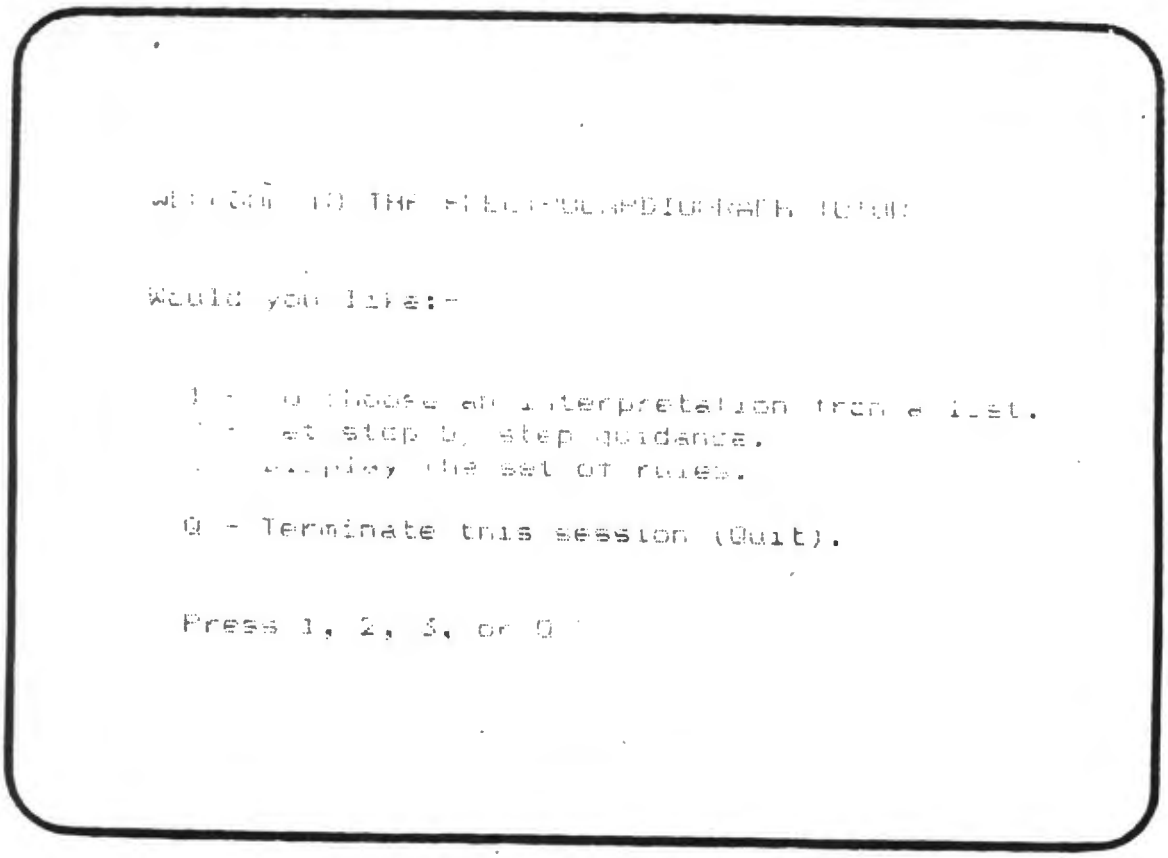

Figure 6.8. Expert system main menu.

If the user chooses to select an interpretation from a list, the list of all possible interpretations for the current knowledge base is provided (Figure 6.9). Once the selection is chosen, the system requests the user to enter the information pertinent to validating the selection chosen by displaying a series of

questions, which the user must answer. An example of one such question is displayed in Figure 6.10. Once all the relevant information has been provided in this manner, a comment is displayed which tells the user whether the interpretation selected is correct or incorrect.

If the step-by-step guidance option is chosen, information is requested in a manner identical to that for the backward chaining method. When enough information has been provided, the program reaches a first conclusion on the interpretation of the electrocardiograph. This conclusion may not be the only possible one, and the user has the option of continuing the search for more possible interpretations. (Figure 6.11). This process can be repeated until all possible interpretations have been found or excluded.

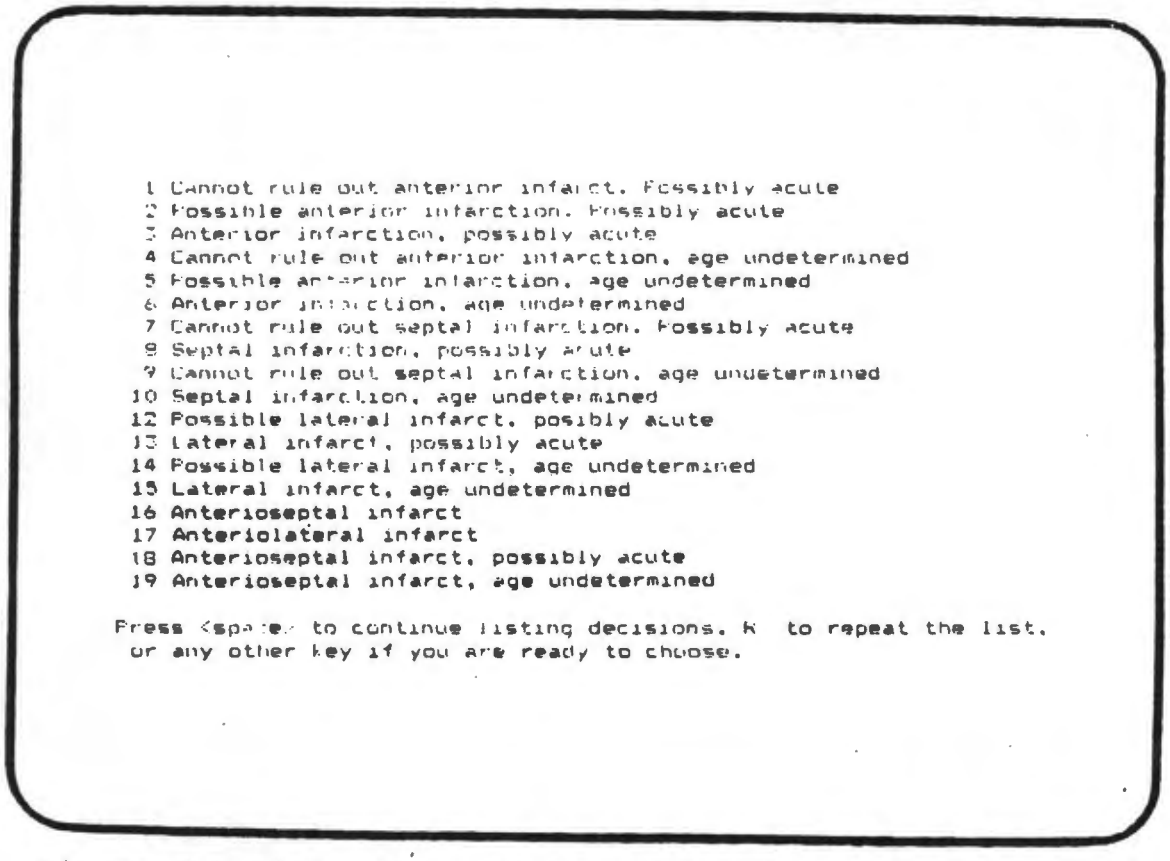

Figure 6.9 List of available interpretations (backward chaining mode.)

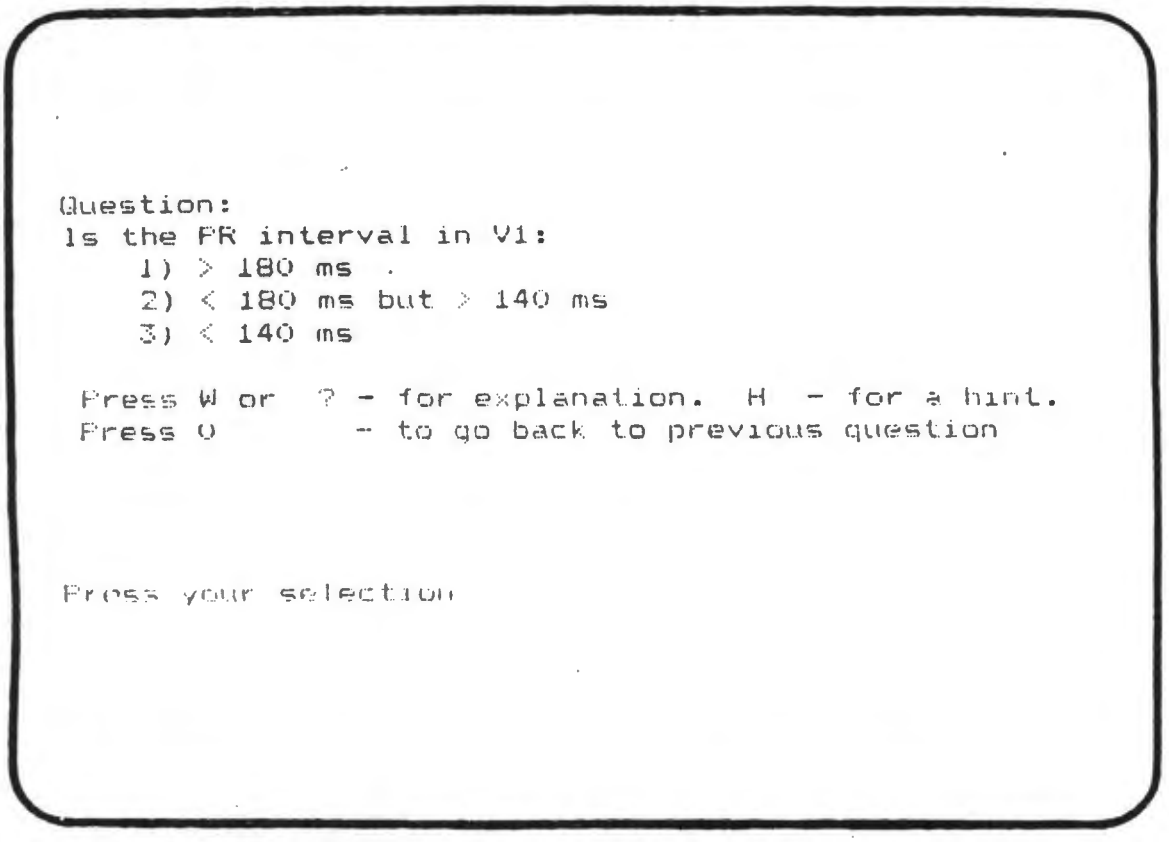

Figure 6.10 An example of a question.

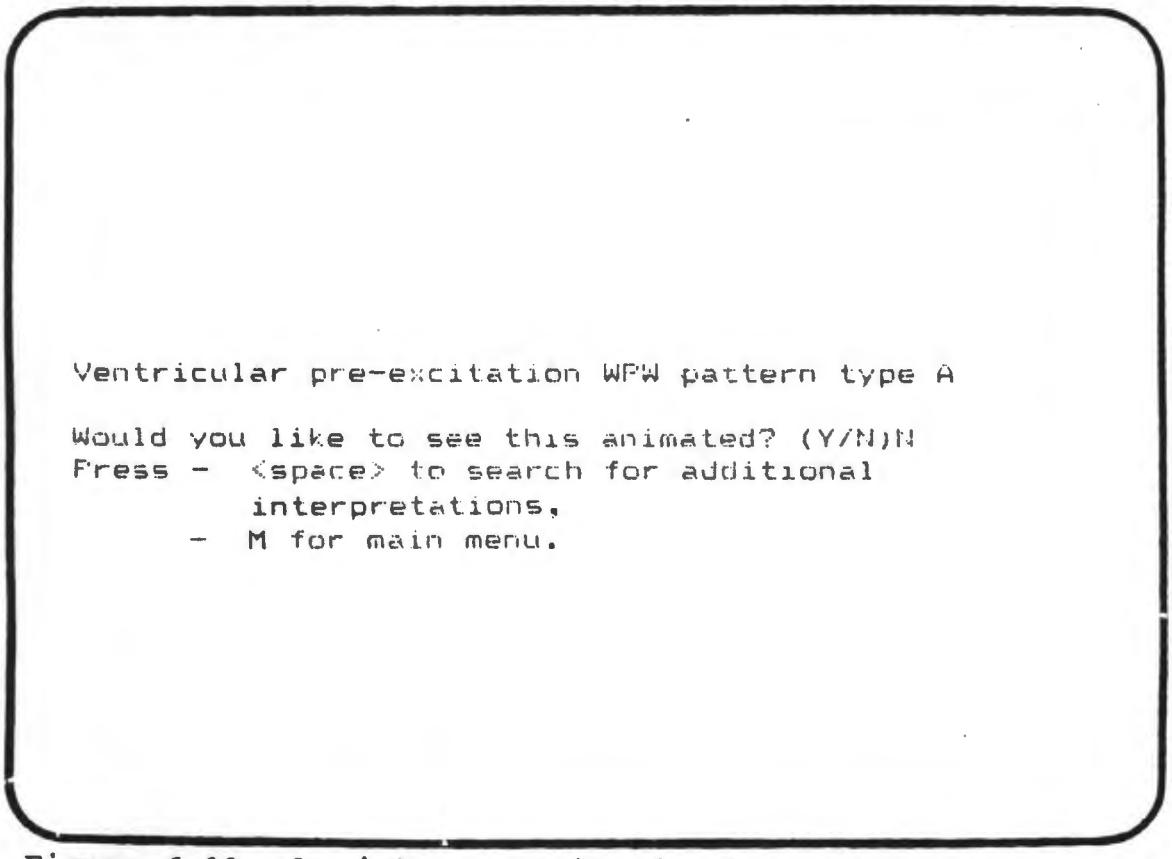

Figure 6.11 An interpretation is found (forward chaining mode).

Once an interpretation has been found, the user has the option of viewing an animated graphic sequence illustrating this abnormatily. A single frame for an animation sequence of twenty-four pictures is reproduced in Figure 6.12, and the entire sequence shown in Figure 6.13.

6.2.4 Flowchart.

The flowchart for this program is contained in appendix I.

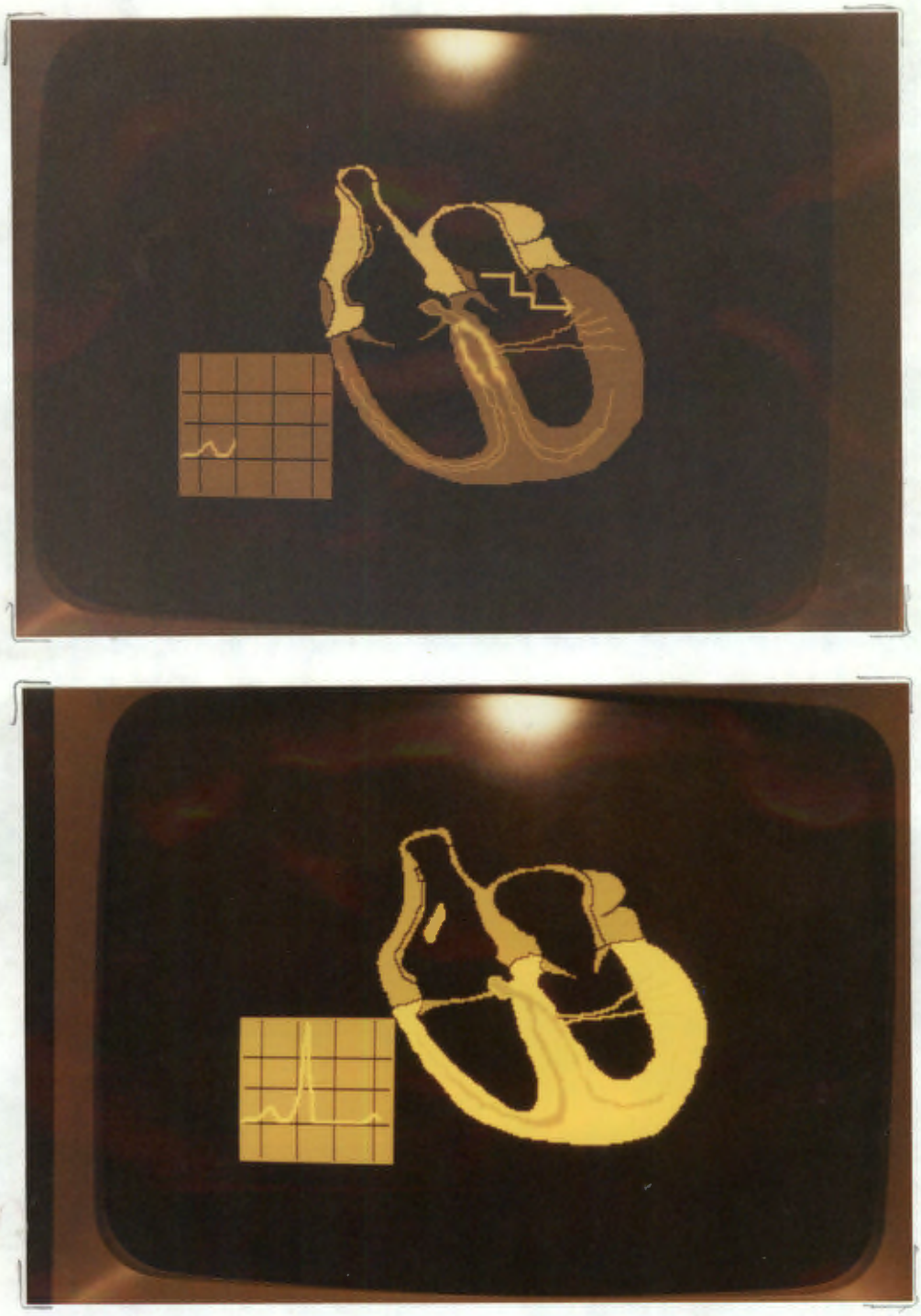

Figure 6.12 : Single frames from the animation sequence.

note:- The relationship between the ECG tracing and the representation of the heart is arbitrary, and does not imply that a particular lead is represented. In this illustration Lead V5 is displayed.

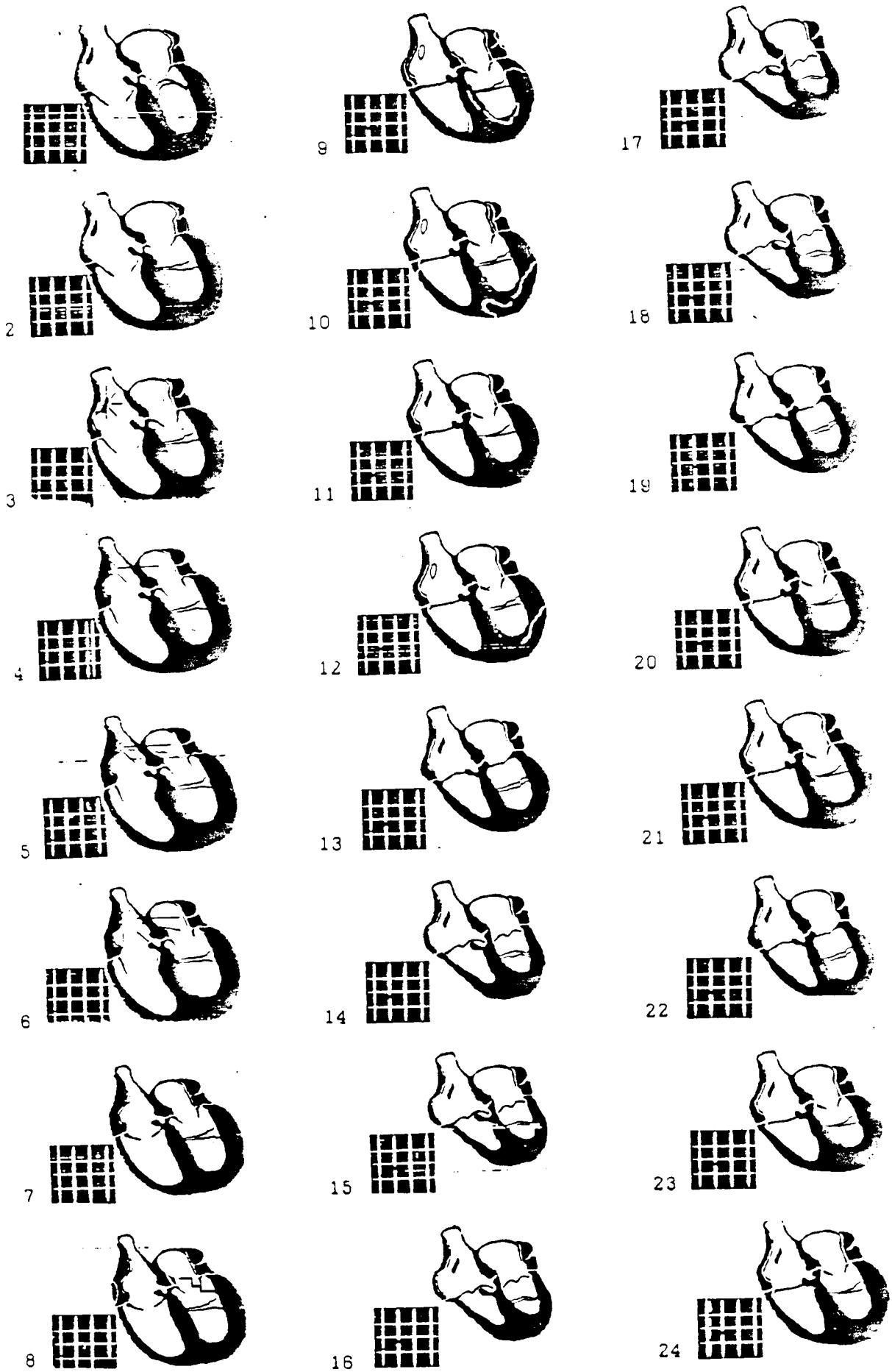

Figure 6.13 : An animation sequence of twenty-four pictures.

6.2.5 Animation sequences.

The aim of the animation sequences is to give an animated graphic representation of each interpretation of the electrocardiograph. When the expert system shell reaches a valid decision (interpretation), the user is asked if he wishes to see this animated.

If the answer is affirmative the animation procedure is entered. The procedure searches for the picture files (on disk) corresponding to the decision number. It will display these files sequentially until no further files in the sequence are found, and then repeat the sequence until interrupted from the keyboard. The animation procedure then returns control to the expert system shell. If no picture files are found corresponding to the decision number, the user is informed of this and control is returned to the shell.

The described method of implementation has many advantages, but also some drawbacks. The animation sequence is selected solely by the decision number. It has no logic showing which knowledge base generated that number. Hence decision number one will produce the same animated graphic sequence for both the Wolff-Parkinson-White or the Infarct knowledge bases, even though the interpretation corresponding to decision number one differs.

The modification to make the animation sequence knowledge base specific can be easily implemented by utilizing knowledge base specific names, i.e calling a picture frame WPW_D1.a, rather than simply D1.a. It is more difficult to incorporate the logic to tell the system that decision number one in one knowledge base requires the same animation sequence as decision number forty-

eight in another, say, without making the writing and modification of knowledge bases a complicated business.

6.3 THE ANIMATION MODULE

Animation requires that a series of images are created (drawn), and then displayed onto the screen in rapid succession. While many commercially available drawing programs exist (e.g. PC-Paintbrush, GEM-Paint) difficulty was experienced in accessing the drawings produced by other packages from the Turbo-pascal environment. The problem is due to the fact that bit-mapped graphics consume considerable amounts of storage space. To improve on this, drawing packages employ data compression algorithms when they store their pictures. I was unable to find documentation which described the algorithm required to decompress the pictures produced by any of the commercially available drawing packages. I therefore wrote my own drawing program, and incorporated it in the animator.

The package consists of two modules:
The picture drawing module.
The data compression module.

6.3.1 The picture drawing (screen painting) module

The picture drawing module allows the user to create a series of drawings on the screen, and then to display them in rapid succession to achieve the effect of animation. The screen-painting module offers none of the user-friendliness found in commercial screen-painting packages, and few of the more advanced functions, but it serve its purpose.

The module offers the following screen painting functions:-

6.3.1.1 Turtle graphics

A triangular marker, approximately 1cm by 1cm, appears on the screen. This is the turtle. It appears in the current drawing colour. It can be moved anywhere on the screen, either by moving along co-ordinates (north, south, east and west) or by rotating to face any direction and then moving forward or backwards. In the drawing mode a line, one pixel wide, will be drawn behind the moving turtle.

6.3.1.2 Circle

The circumference of a circle can be described once the centre and any point on the circumference are marked.

6.3.1.3 Rectangle

A solid rectangle of any available colour can be drawn once diagonally opposite corners are marked. This is intended mainly as a method for painting over (deleting) areas of the picture.

6.3.1.4 Irregular shapes.

Shapes outlined can be filled in any available colour. The turtle must be placed inside the shape to be painted in, and the entire outline must be in the current drawing colour.

6.3.1.5 Copying rectangles.

Any rectangular area of the screen can be copied to any position on the screen. Rectangles cannot be rotated, and will obliterate whatever lies underneath them.

6.3.1.6 Colour.

The animator supports the colour graphics mode of the IBM CG card. Four colours may be used simultaneously on the screen. The animator allows the choice of four palettes:-

Black, Green, Red, Brown.

Black, Cyan, Magenta, Light Gray.

Black, Light green, Bright red, yellow.

Black, Light cyan, light magenta, white.

6.3.1.7 Saving drawings.

The drawing currently on the screen can be saved to a file on disk. Files can have any legal name but must have a single legal ASCII character as a dot extension. Any file with the same name already present on the disk will be over-written.

6.3.1.8 Retrieving drawings.

Picture files stored on disk can be loaded onto the screen for further revision or modification. They can be saved to disk, but will be saved under the same filename. The dot extension, however, can be changed to any legal ASCII character.

6.3.2 Control functions for screen-painting module.

These control functions assume that the keyboard will be used for input. If a mouse is available, it can be used instead of the arrow keys.

Shift <Tab>	Place an old drawing on the screen (load a picture file).
PgUp	Rotate turtle clockwise
PgDn	Rotate turtle anti-clockwise
End or Home	Point turtle Northward
Up Arrow	Point north and move north
Down arrow	Point south and move south
Left arrow	Point west and move west
Right arrow	Point east and move east

Alt <Up arrow>	Move forward
Alt <Down arrow>	Move backwards
F1	Select the first colour for drawing
F2	Select the second colour for drawing
F3	Select the third colour for drawing
F4	Select the fourth colour for drawing
F5	Pen down (drawing mode)
F6	Draw-a-circle mode. In this mode:-

Up arrow	Move turtle north
Down arrow	Move turtle south
Left arrow	Move turtle west
Right arrow	Move turtle east
Ins	Select circle centre
Delete	Select circumference and draw circle, exit circle mode (or exit only if centre not selected).

F7	Clear the screen, lose drawing
F8	Don't display the turtle
F10	Save drawing to disk
Del	Selects first corner and begin paint-rectangle mode. In this mode:

Up arrow	Move turtle north
Down arrow	Move turtle south
Left arrow	Move turtle west

	Right arrow	Move turtle east
	Del	Selects diagonal corner and fills the rectangle with the current drawing colour, then exits paint rectangle.
Ins	Selects first corner and begin copy-rectangle mode:-	
	Up arrow	Move turtle north
	Down arrow	Move turtle south
	Left arrow	Move turtle west
	Right arrow	Move turtle east
	Del	Selects diagonal corner, defining rectangle to copy.
	Ins	Selects new position for the south-east corner and copies the rectangle to this position, then exits this mode.
Shift F1	Fills the shape bounded by the current drawing colour with colour number 1.	
Shift F2	Fills the shape bounded by the current drawing colour with colour number 2.	
Shift F3	Fills the shape bounded by the current drawing colour with colour	

	number 3.
Shift F4	Fills the shape bounded by the current drawing colour with colour number 4.
Shift F5	Pen up. (non-drawing mode)
Shift F6	Reverse colours (invert screen)
Shift F7	Select new palette.
Shift F8	Hide the turtle
Shift F10	Quit

Animation begins automatically once the user leaves the drawing module. The pictures (frames) stored on disk are displayed sequentially. It is assumed (in this version of the animator) that the user chose ".a" to be the extension on the first picture file saved, ".b" for the next, etc. (A single letter extension implies that a maximum of 26 frames can be produced for any one series. This was not a limitation in the developmental stage, as no series exceeded 26 frames. MSDOS allows a maximum of three characters in a dot extension. Should longer series be desirable, a three character extension, which would allow a maximun of over 17 000 frames, could be used.) The choice of the ".a" extension for the first picture works well in practice as the first picture created becomes the first frame of the animation sequence. The first picture is saved with the ".a" extension. The first picture is then modified to become the second frame, which is saved with a ".b" extension, and so on. The animator will continue to display frames until the last frame created in the current session is reached. It will then display the frames backwards and forwards until a key is pressed. The animator then terminates.

6.3.3 Storage and data compression.

There are various strategies which can be employed to transfer an image from the screen to a disk file. Turbo Pascal provides two functions (putpic and getpic) which copy the contents of a rectangular area of the graphics screen into an area of the the data segment of RAM, and vice versa. These functions proved much faster than other methods of transferring data between RAM and the screen (e.g. storing the commands used to generate the picture in the draw screen module, and then re-drawing the screen from the stored command sequence proved exceptionally slow). While the putpic/getpic functions work faster with smaller rectangles than with large rectangles, this difference in speed is small. It is much faster to transfer one 4x4 rectangle than two 2x4 rectangles. It was therefore decided to consider the entire screen as one rectangle, rather than divide it up into smaller rectangles which could be updated selectively.

The amount of RAM required to store any rectangle is given by the formula:

Size = ((width+3)/4) x height x 2 + 6 bytes.

Using the IBM Colour Graphics standard of 320 x 200 pixel resolution, the above formula gives 32 kb. This is fairly large. If the animator could display pictures at the motion picture rate of 24 frames per second, a 20 megabyte hard disk would be able to hold just 26 seconds of movie, but this may be improved upon by data compression.

6.3.3.1 Mode of operation.

The data compression module is a stand-alone module, and does not interface with the animation module. It functions by reading a

series of picture files, and writes another series of the same files in compressed form.

The data compression module begins by confronting the user with a prompt asking for the name of a picture file. Once this is provided it asks the user to indicate the beginning of the series. It reads a single ASCII character, which becomes the dot extension for the picture file name requested earlier. It then reads in this file. If the file is not found the program terminates.

It then reads in the picture file with the same name but the next dot extension, i.e. if the name of the picture file is "D1", and the series began at "a", the program will first read in D1.a, then D1.b. The compressed version of D1.b will then be written to disk under the name D1c.b.

D1.c will then be accessed, and a compressed version D1c.c created. The process will continue until no further files in the series are found.

A verification procedure will then display the picture contained in the first (decompressed) file on the screen - in this case picture D1.a. It will then read in the compressed files, beginning at D1c.b in our example, and decompress this to display picture D1c.b. If this is identical to picture D1.b then all is well. Thus a series of pictures will be displayed until no further pictures in the series are found. The verification procedure will be repeated until a key is pressed. Thus an animation sequence is displayed on the screen, which is running as fast as is possible.

In practice animation using compressed pictures ran at roughly the same speed as animation using decompressed pictures, at 2 to 3 frames per second. This is because:-

a) Less data was read off disk. Since disk access is slow, even off a hard disk, this represents a time saving.

b) The compressed picture has to be decompressed before it can be displayed on screen, and the required computation uses up the time saved in accessing less data.

6.3.3.2 Data compression algorithm.

The delta data compression algorithm is used. In this implementation the first picture of the series is considered the "root" frame. The second frame is compared to the first frame. Any differences between frames are recorded, and this record becomes the compressed version of the second frame. The second frame is then compared to the third frame, and the differences stored as the compressed version of the third frame, and so on.

6.3.3.3 Efficacy of data compression.

The following table gives the size, in bytes, of the picture files in the D1 animation series from B to Z.

Picture	Size	Size after compression.
D1.B	32768	512
D1.C	32768	128
D1.D	32768	3456
D1.E	32768	3328
D1.F	32768	1280
D1.G	32768	2048
D1.H	32768	512
D1.I	32768	3584
D1.J	32768	1152
D1.K	32768	7040
D1.L	32768	5248
D1.M	32768	4992
D1.N	32768	3072
D1.O	32768	5888
D1.P	32768	6272
D1.Q	32768	2432
D1.R	32768	128
D1.S	32768	128
D1.T	32768	128
D1.U	32768	2432
D1.V	32768	9984
D1.W	32768	3840
D1.X	32768	6656
D1.Y	32768	8320
D1.Z	32768	9856
Total	850 k	120 k

Table 6.1 : Sizes of picture files before and after compression.

Since the "root" picture is not included, 33k must be added to both totals. This gives

Decompressed 883k

Compressed 153k

Compression ratio 17.3%

The above compression ratio offers significant savings in data storage. With compressed files it is possible to store the entire sequence of 26 frames on a single floppy disk, and this simplifies transferring sequences from one computer to another.

A second data compression algorithm, the "string" algorithm, was

also investigated. If the picture file is considered as a one dimensional array, it can be stored by noting the type of pixel and the number of times it recurred, i.e. the length of the string of that pixel type. This would not offer much reduction in size if the pixel type changed often, but as the pictures used in this project consisted largely of areas of a single colour, the algorithm may offer significant data compression.

The algorithm has two advantages over the delta data compression algorithm. As each picture is compressed independently of preceding pictures, any pollution of data would affect only the one picture, not the entire series that followed. The string algorithm is also much simpler to code.

In practice, it was found that the picture compression ratio using the string algorithm was, on average, two to three times worse than that of the delta compression algorithm.

6.3.4 Flowchart.

The flowchart for the program is contained in appendix I.

CHAPTER 7

CONCLUSION AND RECOMMENDATIONS

7.1 CONCLUSION

A new computer interpretative electrocardiograph is described which offers explanatory and teaching functions, and so can be used as a tutoring system as well as an interpretative system. Explanatory functions consist of:

a) The display of the electrocardiograph trace, with facilities for reviewing and overriding the system's identification of individual waves.
b) The display of the interpretation in the form of an animated graphic.
c) The trace of the logic which led to a particular conclusion.

The prototype system developed has demonstrated that all these three objectives can be achieved. The system described requires superior graphics, and the IBM PC used was not equal to the task. The IBM PC also lacked the speed required of an interactive system.

The prototype system was not interfaced to the required ECG amplifier, and a suitable pattern-recognition algorithm was not implemented. The knowledge bases written for the system only covered a small part of diagnostic electrocardiology.

Past projects to develop computerised ECG analysis systems (e.g. the ECAN project) required considerable funding over a number of years. While much of this development is in the public domain, and so need not be repeated, development of the ECG Tutor into a viable system would still require a large effort by a team with

skills in cardiology, computer science, computer animation and signal processing. Such an effort is probably beyond the capabilities of a small department with limited resources.

7.2 RECOMMENDATIONS

While the development of the ECG Tutor, capable of recording and interpreting ECG traces from a patient in a manner comparable with existing ECG analysis systems, is a fairly large project, it can be broken into more manageable stages.

The first stage involves the development of a system solely for the purpose of computer aided instruction. I suggest the following:

a) As the system is required for instruction purposes only, it is not necessary to be able to record ECG tracings. The "front end" of the system, i.e. the ECG amplifier, A-D conversion card, and signal processing and pattern-recognition modules, as well as the expert system module, are not required.
b) ECG tracings be entered into the system, not as an electric signal, but in graphic form. This entails first obtaining a high quality ECG tracing on a paper. A page scanner is then used to enter the ECG into the system.
c) A database of scanned ECG tracings be built up in this manner. Each ECG trace in the database should be associated with an interpretation, a series of questions particular to that ECG, and each question must be associated with a correct answer.
d) A second database, consisting of animation sequences, must be created. An animation sequence must exist for each interpretation in the database of ECG tracings.
e) The system must have the ability to allow the student to

select a scanned ECG trace from the database. The ECG trace selected must then be displayed on the screen. The questions associated with that ECG then appear on the screen in sequence. The student is required to enter an answer to each question, and if the student's answer corresponds to the correct answer (which already exists in the database), the next question in the series particular to the ECG selected is asked.

f) If the student answers all the questions correctly, the interpretation (which also already exists in the database) is displayed, and the animation sequence associated with that interpretation shown.

This system requires little computational power. What is required are high quality graphics and animated graphics, and enough on-line storage to hold the databases. The Commodore Amiga (developed primarily as a games-playing machine) is one machine which is relatively cheap and can support animated graphics. Software for the creation of animation is also available for the Amiga.

Should such a system prove popular as a teaching tool, consideration can be given to the development of a system capable of recording and interpreting ECG signals.

REFERENCES

ADLASSNIG KP, KOLARZ G.
1986
Representation and Semiautomatic Acquisition of Medical Knowledge in CADIAG-1 and CADIAG-2
Computers and Biomedical Research, 19:63-79

ANONYMOUS
1984
Why can't a doctor be more like a computer?
(Editorial) The Economist 1984 Dec.1: 69-71.

ANONYMOUS
1985
Recommendations for measurements standards in quantitative electrocardiography.
(Editorial) European Heart Journal, 6: 815-825.

BARR RC, SPACH MS.
1977
Sampling rates required for digital recording of intracellular and extracellular cardiac potentials.
Circulation 55(1):40-48.

BERNARD P, CHAITMAN BR, SCHOLL JM, VAL PG, CHABOT M.
1983
Comparative diagnostic performance of the Telemed computer ECG program.
Journal of Electrocardiology, 16(1):97-102.

BERSON AS, WOJICK JM, PIPBERGER HV.
1977
Precision requirements for electrocardiographic measurements computed automatically.
IEEE Transactions on Biomedical Engineering BME, 24(4):382-385.

BLUM BI
1985
Artificial intelligence and medical informatics.
Journal of Clinical Engineering, 10(2):109-120.

BODEN M.
1977
Artificial Intelligence and Natural Man.
Harvester Press Ltd, Brighton, England.

BONNER RE, CREVASSE L, FERRER MI, GREENFIELD JC.
1978
A new computer program for comparative analysis of serial scalar electrocardiograms: description and performance of the 1976 IBM program.
Computers and Biomedical Research, 11:103-118.

BONNER RE, CREVASSE L, FERRER MI, GREENFIELD JC.
1983
The influence of editing on the performance of a computer program for serial comparison of electrocardiograms.
Journal of Electrocardiology, 16(2):181-189.

BONNER RE, JACKSON LK, HOOPER JK, SMITH CH, HSIEH KC.
1979
A method for testing performance in Electrocardiographic interpretation.
Proceedings of the 1979 Engineering Foundation Conference on Computer Interpretation of the ECG IV, pp 128-151.

CACERES CA.
1973
The case against electrocardiographic automation.
Computer, July 1973 :15-21.

CACERES CA.
1976
Limitations of the Computer in Electrocardiographic Interpretation.
The American Journal of Cardiology, September pp 362-376.

CERUTTI S, GATTI E, MASCIADRI L.
1980
Digital filtering and baseline-drift correction algorithms in the computerized pre-processing of ECG tracings.
pp. 231-235 In: Lindberg/Kaihara eds. MEDINFO 80. North-Holland Publishing Company.

CLINICAL MICRODATA SYSTEMS,INC.
1985
Introducing the multivariate digital cardiograph.
Clinical Microdata Systems,Inc. Melbourne Florida. (Brochure.)

COOMBS MJ (ed).
1984
Developments in Expert Systems.
Academic Press, London.

CRITCHFIELD GC, WILLARD KE, CONNELLY DP.
1986
Probabilistic sensitivity analysis methods for general decision models.
Computers and Biomedical Research, 19:254-265.

DOUE JC, VALLANCE AG.
1985
Computer-aided ECG analysis.
Hewlett-Packard Journal, 36(9):29-34.

DRAZEN E.
1980
How to Decide Among Alternative Computerised ECG Systems.
Arther D. Little, Inc. Cambridge, Mass.

DUDECK J.
1980
The impact of new technology on automatic ECG processing.
pp.313-319 In: Rienhoff O, Abrams ME. eds. The Computer in the doctor's office. North-Holland Publishing Company.

FORSYTH R, NAYLOR C.
1985
The Hitch-Hikers Guide to Artificial Intelligence.
Chapman and Hall Ltd, London.

GOETOWSKI CR.
1977
The Telemed System.
in:Trends in Computer-processed Electrocardiology, van Bemmel JH and Williams JL (ed) pp 207-210, North-Holland.

GOLD RG.
1985
Do we need a new standard for electrocardiographs?
British Heart Journal, 54:119-120.

GOLDING J
1986
Cardiac rhythms and arrhythmias: a teaching program
Computer Methods and Programs in Biomedicine, 23:331-336

HARMON P, KING D.
1985
Aritificial Intelligence in Business
John Wiley and Sons, Inc. New York.

HEWLETT-PACKARD COMPANY.
1984.
PageWriter interpretive cardiograph Model 4760AI.
Technical Information brochure from Hewlett-Packard.

HORBAR JD.
1985
A computer simulation of medical decision strategy performance.
Computer and Biomedical Research, 18: 563-575.

JOSIN G.
1987
Neural network Heuristics: Three heuristic algorithms that learn from experience.
Byte November 1987: 183-192.

LAKS MM.
1980
Computers in Cardiology. Characteristic Features Making It a Useful Adjunct to the Electrocardiographer.
In: Computer Systems for the processing of Diagnostic Electrocardiograms, IEEE Computer Society Press, New York.

MILLER PL.
1986
The evaluation of artificial intelligence systems in medicine.
Computer Methods and Programs in Biomedicine, 22:5-11.

MILLER RA, POPLE HE, MYERS JD.
1982
INTERNIST-I, an experimental computer-based diagnostic consultant for general internal medicine.
The New England Journal of Medicine, 307(8):468-476.

MILLIKEN JA, HENDERSON J.
1978
Assessment of the technical quality of electrocardiograms.
CMA Journal, 119:327-333.

MILLIKEN JA, PIPBERGER H, PIPBERGER HV. et al.
1983
The impact of an ECG computer analysis program on the cardiologist's interpretation. A cooperative study.
Jiurnal of Electrocardiology, 16(2):141-149.

NAYLOR C.
1983
Build your own Expert System
Sigma Technical Press, U.K.

PIPBERGER HV.
1977.
The ECG computer analysis system developed in the U.S. Veterans Administration.
In: Trends in Computer-processing Electrocardiograms pp 125-132, North-Holland.

POPPL SJ.
1980
Development of microcomputers (Dedicated ECG-systems).
pp.299-311 In: RIENHOFF O, ABRAMS ME, eds. The computer in the doctor's office. North-Holland Publishing Company.

PRYOR TA, DRAZEN E, LAKS M.
1980
Computer systems for the processing of diagnostic electrocardiograms.
IEEE Computer Society Press, New York.

QUAGLINI S, STEFANELLI M.
1986
ANEMIA: An Expert Consulting System
Computers and Biomedical Research, 19:13-27

RENNELS GD, SHORTLIFFE EH.
1987
Advanced Computing for Medicine
Scientific American, 257(4):146-153

RICHARDS B, BRAY CL, JEFFERY C, KHADR N.
1980
An assessment of the accuracy of the Bonner/IBM ECG analysis program when compared with autopsy evidence.
pp.244-248 In: Lindberg/Kaihara eds. MEDINFO 80. North-Holland Publishing Company.

SCHAMROTH L.
1980
The Wolff-Parkinson-White syndrome.
Postgraduate Medical Services, Cardiovascular 3(1):1-35.

SHEFFIELD LT, PRINEAS R, COHEN HC, SCHOENBERG A, FROELICHER V.
1978
Quality of electrocardiographic records. Task Force 2.
American Journal of Cardiology, 41:146-157.

SHRIDHAR M, STEVENS MF.
1979
Analysis of ECG data, for data compression.
International Journal of Bio-Medical Computing, 10:113-128

SPIEGELHALTER DJ, KNILL-JONES RP
1984
Statistical and knowledge-based approaches to clinical decision-support systems, with an application in gastroenterology.
Journal of the Royal Statistical Society Series A (General), 147(1):35-77.

SPIEGELHALTER DJ.
1984a
Computer aided decision making in medicine.
British Medical Journal, 289:567-568.

TALMON JL, HASMAN A.
1980
An evaluation of algorithms for QRS-typification.
pp.249-253 In: Lindberg/Kaihara eds. MEDINFO 80. North-Holland Publishing Company.

TAYLER DI, VINCENT R.
1983
Signal Distortion in the Electrocardiogram Due to Inadequate Phase Response.
IEEE Transactions on Biomedical Engineering, 30(6):352-356.

TORASSO P
1985
Knowledge based expert systems for medical diagnosis.
Statistics in Medicine, 4:317-325

TUINSTRA CL, RAUTAHARJ PM, PRINEAS RJ, DUISTERHOUT JS.
1982
The performance of three visual coding procedures and three computer programs in classification of electrocardiograms according to the Minnesota Code.
J. Electrocardiology, 15(4):345-349.

WEISS SM, GALEN RS.
1986
An expert system for diagnosis of myocardial infarction.
pp.219-221 In: Salamon R, Blum B, Jorgensen M, eds. MEDINFO 86 North-Holland: Elsevier.

YU BC, LIU CS, LEE M, CHEN CY, CHIANG BN.
1985
A nonlinear digital filter for cardiac QRS complex detection.
Journal of Clinical Engineering, 10(3):193-201.

ADDITIONAL SELECTED READING

ADLASSNIG KP, KOLARZ G, SCHEITHAUER W, EFFENBERGER H, GRABNER G.
1985
CADIAG: Approaches to computer-assisted medical diagnosis.
Computurs in Biology and Medicine, 15(5):315-335.

ADLASSNIG KP, KOLARZ G, SCHEITHAUER W.
1985a
Present state of the medical expert system CADIAG-2.
Methods of Information in Medicine, 24(1):13-20

ADLASSNIG KP, KOLARZ G.
1986
Representation and semiautomatic acquisition of medical knowledge in CADIAG-1 and CARDIAG-2.
Computers and Biomedical Research, 19:63-79.

ADLASSNIG KP.
1986
Fuzzy Set theory in medical diagnosis.
IEEE Transactions on Systems, Man and Cybernetics, SMC-16(2):260-265.

AMSTERDAM J.
1985
An analysis of sorts. How to choose one sorting algorithm over another.
Byte, Sept.1985 105-112

ANONYMOUS
1985
Recommendations for measurements standards in quantitative electrocardiography.
(Editorial) European Heart Journal, 6: 815-825.

ANONYMOUS
1986
Expert systems and Artificial intelligence.
Nixdorf Computers (South Africa) (promotional literature)

ANONYMOUS
1986a
Shelf-based expert system.
Understanding Computers, Time-Life Books, Chicago (promotional literature)

ANONYMOUS
1987
Three Neural-network programs
(Editorial) Byte, November 1987: 45.

ANONYMOUS.
1987
Turn your AT into a Neurocomputer
(Editorial) Byte, November 1987: 46.

BAILEY JJ, HORSTON M, ITSCOITZ SB.
1974
A Method for Evaluation Computer Programs for Electrocardiograph Interpretation.
Circulation, 50: 88-93.

BALDA RA, DILLER G, DEARDOFF E, DOUE J, HSIEH P.
1977
The HP ECG Analysis Program.
pp 197-205, In: Trends in Computer-processed Electrocardiograms, North-Holland.

BANTA RA, DORWARD PH, SCAMPINI SA.
1985
New cardiograph family with ECG analysis capability.
Hewlett-Packard Journal 36(9):23-28.

BERNARD P, CHAITMAN BR, SCHOLL JM, VAL PG, CHABOT M.
1983
Comparative diagnostic performance of the Telemed computer ECG program.
Journal of Electrocardiology, 16(1):97-102.

BLACKBURN H, KEYS A, SIMONSON E, RATAHARJU P, PUNSAR S.
1960
The electrocardiogram in population studies.
Circulation, 21:1160-1175.

BORROW DG, STEFIK MJ.
1986
Perspectives on artificial intellengence programming.
Science, 231:951-967.

BYLES T.
1984
Computer animation shows inner workings of the human body.
IEEE Computer Graphics and Applications, Dec 1984 pp 68-69

CERUTTI EG, MASCIADRI L.
1982
An automatic procedure for the pre-processing ECG/VCG signals.
International Journal Bio-Medical Computing, 13:329-341.

CLEMONS EK, GREENFIELD AJ.
1985
The SAGE System Architecture: A system for the Rapid Development of Graphics Interface for Decision Support.
IEEE Computer Graphics and Applications Nov 1985 pp 38-49.

COLEMAN JD, BOLTON MP.
1979
Microprocessor detection of electrocardiogram R-waves.
Journal of Medical Engineering and Technology, 3(5):235-241.

CRECINE JP.
1986
The next generation of personal computers.
Science, 231:935-950.

D'AMBROSIO B.
1985
Expert systems - myth or reality?
Byte, January 1985: 275-282.

DEERING MF
1985
Architectures for AI. Hardware and software for efficient processing.
Byte, April 1985, 193-206.

EDDY DM, CLANTON CH.
1982
The art of diagnosis. Solving the clinicopathological exercise.
The New England Journal of Medicine, 306(21):1263-1268.

EVANS AL, SMITH DC, WATTS MP.
1984
Microprocessor-controlled signal generator for the functional testing of electrocardiographs.
Medical & Biological Engineering & Computing 22:468-470.

FEINER S.
1985
APEX: An Experiment in the Automated Creation of Pictorial Explanations.
IEEE Computer Graphics and Applications, Nov 1985: 29-37

FISHHOF TJ.
1982
Electrocardiographic diagnosis using digital differentiation.
International Journal of Bio-Medical Computing, 13:441-446.

FOLEY JD, WALLACE VL, CHAN P.
1984
The Human Factors of Computer Graphics Interaction Techniques.
IEEE Computer Graphics and Applications, Nov 198: 13-48

FONTAINE D, LE BEUX P.
1986
An expert system for computer assisted instruction simulations.
pp.878-882 In: Salamon R, Blum B, Jorgensen M, eds. MEDINFO 86.
North-Holland: Elsevier.

FUGLEBERG S.
1986
Computer-Assisted Diagnosis of Acute Azotaemia.
Computers and Biomedical Research, 19:103-105

HELDER JC, SCHRAM PH, VERWEIJ H, ROBLES DE MEDINA EO, MEIJLER FL.
1980
Integrated processing of electrocardiograms in a hospital information system. (Abstract).
p.274 In: Lindberg/Kaihara eds. MEDINFO 80. IFIP, North-Holland Publishing Company.

HEWLETT-PACKARD COMPANY
1976
A compendium on automated arrhythmia detection.
pp.1-37 Hewlett Packard brochure 5952-5333 6/78

HEWLETT PACKARD COMPANY
1979
HP ECG analysis program. Physician's guide.
Hewlett Packard brochure 05600-91911 1/80.

HEWLETT-PACKARD COMPANY
1984
PageWriter Measurements Cardiograph. Model 4760AM
Technical Information pamphlet from Hewlett-Packard.

HEWLETT-PACKARD COMPANY
1984a
PageWriter Alphanumeric Cardiograph Model 4760A.
Technical Information brochure from Hulett-Packard.

HEWLETT-PACKARD COMPANY.
1984b.
PageWriter interpretive cardiograph Model 4760AI.
Technical Information brochure from Hewlett-Packard.

HEWLETT PACKARD COMPANY.
1984c.
Plug-in application modules for PageWriter cardiographs. Model 47600 series.
Technical Information brochure from Hewlett-Packard.

HEWLETT-PACKARD COMPANY.
1985
The PageReader. A collection of electrocardiograms recorded by Hewlett-Packard's PageWriter Cardiograph.
Hewlett-Packard Andover Division, Massachusetts,
brochure 5953-4941 2/81.

HULTING J, BLOMQVIST P, NYGARDS ME.
1977
Computer-based ECG analysis in acute myocardial infarction.
Acta Medica Scandinavica, 201:439-447.

IVEY SL, BERGERON BP
1986
A computer simulation for teaching clinical cardiology.
p.1152 In: Salamon R, Blum B, Jorgensen M, eds. MEDINFO 86.
North-Holland: Elsevier.

JACOB VS, GAULTNEY LD, SALVENDY G.
1986
Strategies and biases in human decision-making and their implications for expert systems.
Behaviour and Information Technology, 5(2):119-140.

JESSE MJ.
1985.
Advances in cardiology and escalating costs to the patient.
Circulation, 17(3):423.

JOHNSON AT.
1985
Microcomputer program for the design of digital filters.
Computer Methods and Programs in Biomedicine 21:203-210.

KORS JA, TALMON JL, VAN BEMMEL JH
1986
Computerized ECG interpretation: A knowledge engineering view.
pp.68-70 In: Salamon R, Blum B, Jorgensen M eds. MEDINFO 86
North-Holland: Elsevier.

KOWALSKI R.
1982
Logic for Problem Solving.
Elsevier North Holland, Inc. New York.

LENFANT C, ROTH CA.
1985
Advances in cardiology and escalating costs to the patient: a view from the government.
Circulation 17(3):424-428.

LOVELL N, CELLER B.
1986
CARDIOSYS: An ECG Analysis Package.
Engineering and the Physical Sciences in Medicine Conference, Sydney August 1986.

MACFARLANE PW, MACFARLANE DK, PODOLSKI M, LAWRIE TDV.
1984
The ECG analysis program for the MINGOCARE system.
Electro Medica, 52:126-136.

MASARIE FE, MILLER RA, MYERS JD.
1985
INTERNIST-I Properties: Representing common sence and good medical practice in a computerized medical knowledge base.
Computers and Biomedical Research, 18:458-479.

MATHIASSEN L, MUNK-MADSEN A.
1986
Formalizations in systems development.
Behaviour and Information Technology, 5(2):145-155.

MEIJLER FL, ROBLES DE MEDINA EO, HELDER JC.
1980
Future of computerised electrocardiography.
British Heart Journal, 44:1-4.

MICHAELSEN RH, MICHIE D, BOULANGER A.
1985
The technology of expert systems. Transplanting expert knowledge to machines.
Byte, April 1985: 303-312.

O'KANE KC
1986
An expert systems facility for MUMPS.
Computers in Biololgy and Medicine, 16(3):205-213.

RAUTAHARJU PM, OKAJIMA M.
1980
Computerized electrocardiography: old problems and new challenges.
pp.228-230 In: Lindberg/Kaihara eds. MEDINFO 1980. Pub. IFIP,
North-Holland Pub. Co.

RICHER MH, CLANCEY WJ.
1985
GUIDON-WATCH: A graphic interface for viewing a knowledge-based system.
IEEE Computer Graphics and Applications Nov 1985 pp 51-64

RICHER MH.
1986
An e luation of expert system development tools.
Expe. . Systems, 3(3):166-182.

RIOS J, SANDQUIST F, RAMSETH D, STRATBUCKER R, DRAZEN E, HANMER J.
1977
Cost effectiveness of the Electrocardiogram.
The American Journal of Cardiology, (April): 65-73.

RUBEL P, VARROT M, MORLET D, BLOCH JR, ARNAUD P, FORLINI MC
1980
The Lyon microcomputer based portable ECG processing system.
p.1345 In: Lindberg DAB, Kaihara S eds. MEDINFO 80. North-Holland Elsevier.

SHAHEIN HI.
1983
Computers in health education.
Computer Programs in Biomedicine, 17:213-224.

SILVA J, HAGAN AD.
1974
An economic analysis of an automated system for interpreting electrocardiograms.
Computers and Electrical Engineering, 1:559-571.

SIX P, CHAUVET G.
1986
An efficient structure for a medical information database in a general hospital.
International Journal of Bio-Medical Computing, 18:213-227.

SPELLER GJ, BRANDON JA.
1986
Ethical dilemmas constraining the use of expert systems.
Behaviour and Information Technology, 5(2):141-143.

STERNBERG RJ.
1985
Human intellengence: The model is the message.
Science, 230(4730):1111-1118.

TALMON JL, KORS JA, VAN BEMMEL JH.
1986
Algorithms for the detection of events in electrocardiograms.
Computer Methods and Programs in Biomedicine, 22:149-161.

TAYLER DI, VINCENT R.
1985
Artefactual ST segment abnormalities due to electrocardiograph design.
British Heart Journal, 54:121-128.

THOMAS RA, BOWYER AF.
1986
Development of electrocardiographic teaching materials using an MC68000-based, interactive graphics microcomputer.
Computer Methods and Programs in Biomedicine, 22:87-91.

THOMPSON BA, THOMPSON WA.
1985
Inside an expert system. From index cards to Pascal program.
Byte, April 1985: 315-330.

THOMPSON T
1986
The Commodore Amiga.
Byte, November 1986: 231-237.

TRIM JC, KORS JA.
1986
A personal ECG analysis system.
p.1159 In: Salamon R, Blum B, Jorgensen M, eds. MEDINFO 86.
North-Holland: Elsevier.

VOSE GM
1987
Introduction: Heuristic Algorithms
Byte, November 1987: 148.

YAZDANI M.
1986
Intelligent tutoring systems: An overview.
Expert Systems, 3(3):154-162.

ZYWIETZ C, & WORKING GROUP 3 OF IFIP TC4.
1980
Procedures and recommendations for computerized ECG analysis: report on a TC4 WG4.3 working document.
pp.269-273 In: Lindberg/Kaihara eds. MEDINFO 80. North-Holland Publishing Company.

APPENDIX I

PROGRAM FLOWCHARTS

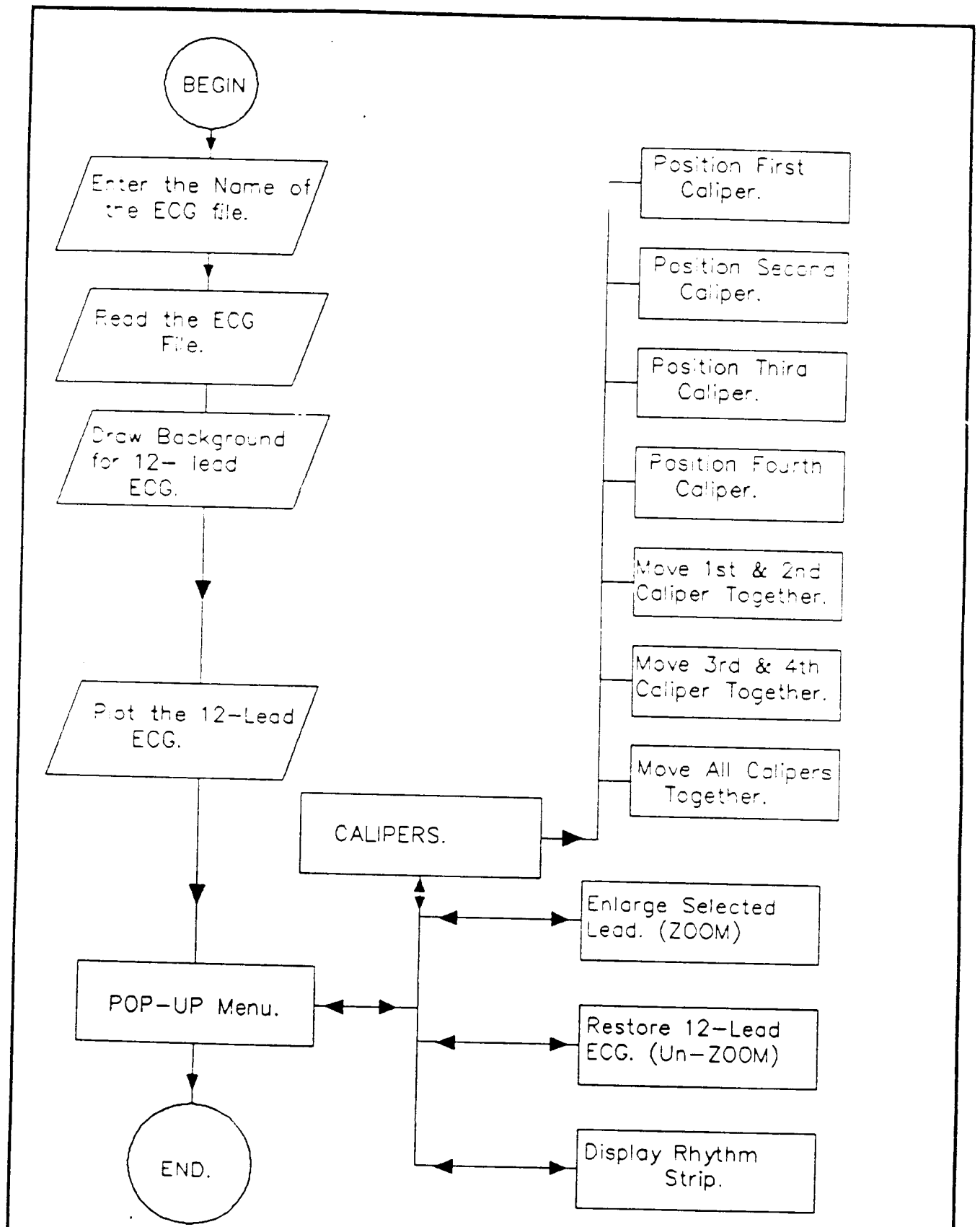

FIG. A1.1: ECG Display Module Flowchart.

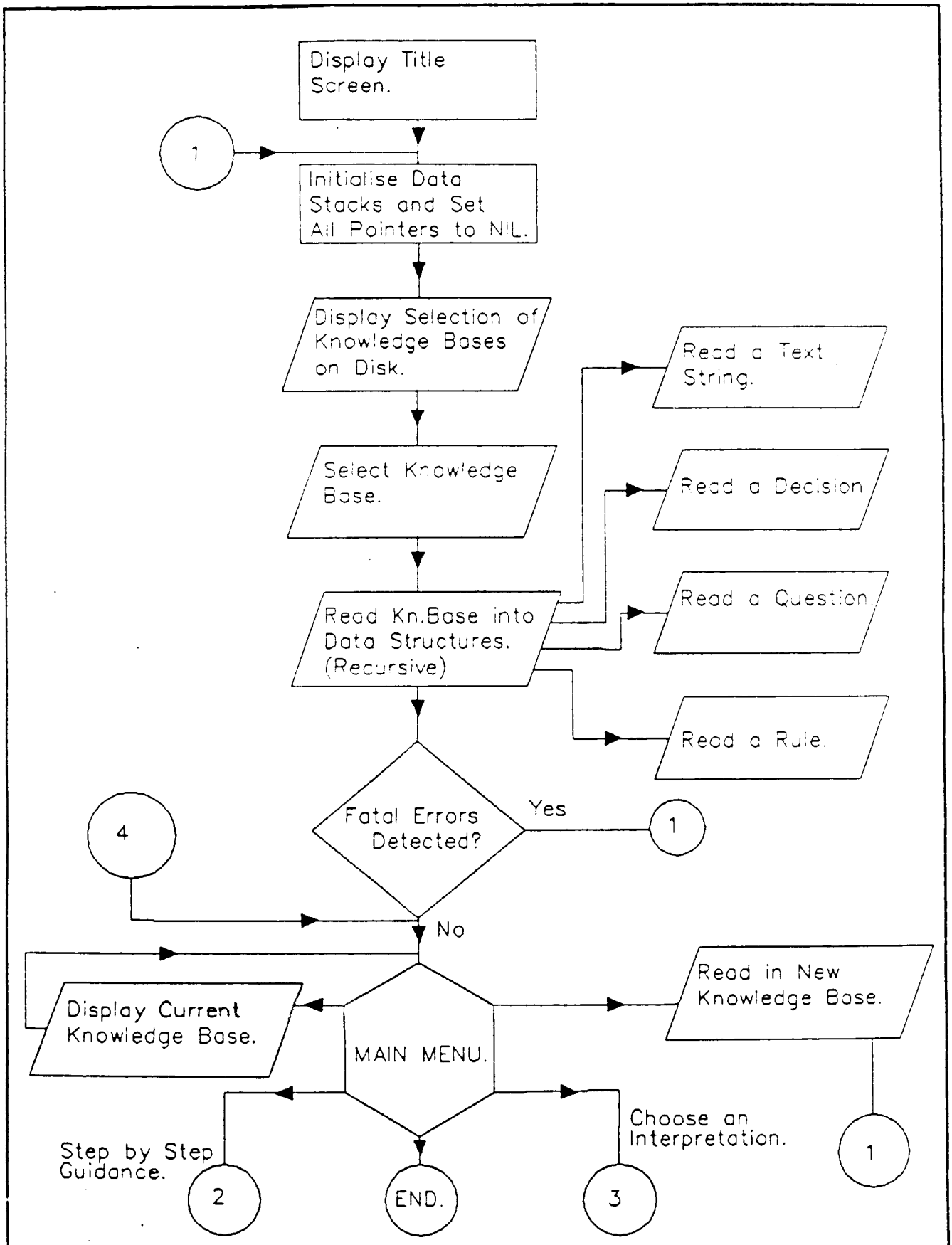

FIG A1.2: Expert System Shell Flowchart.

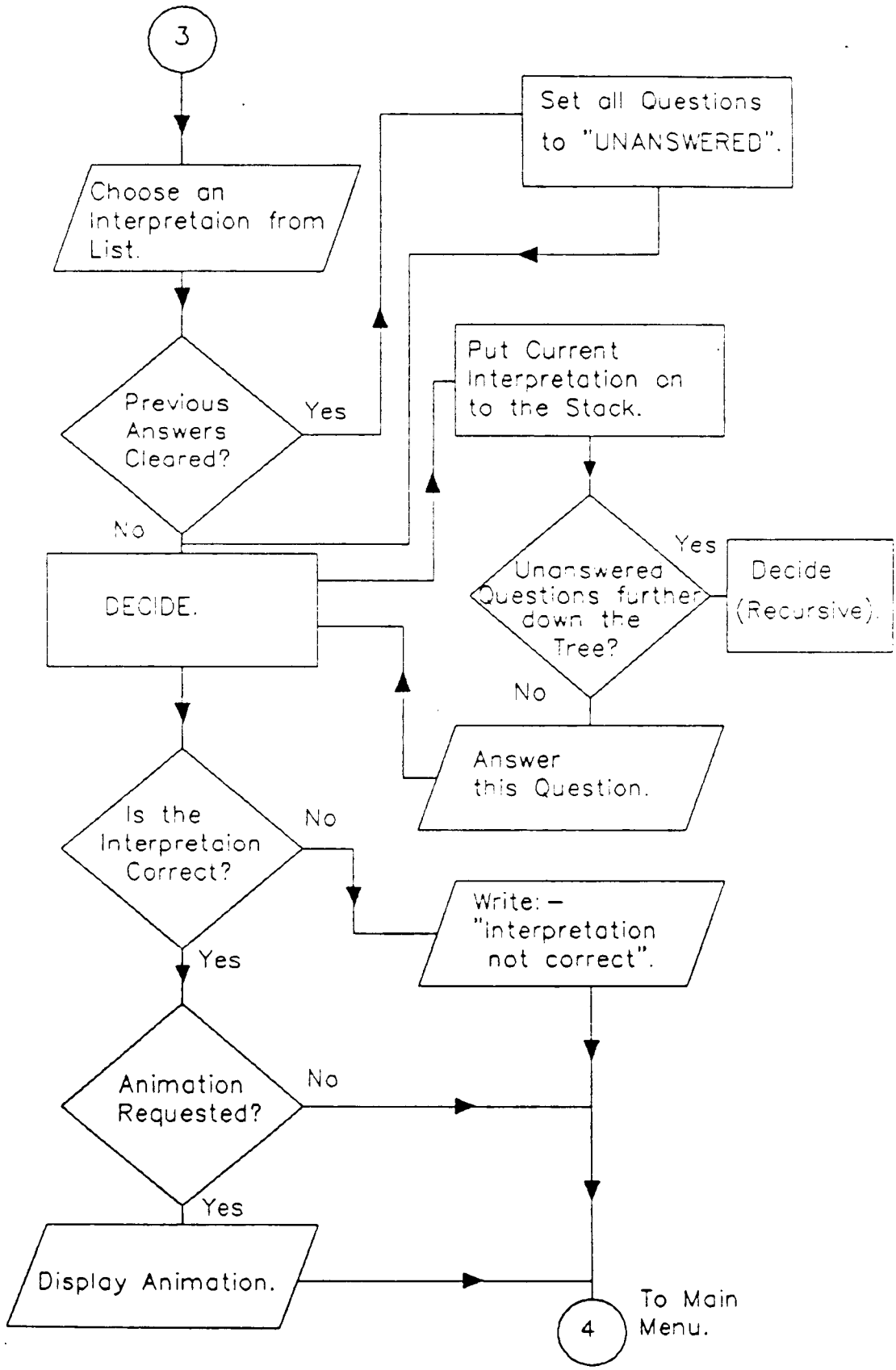

FIG A1.3 Expert System Flowchart (cont.1)

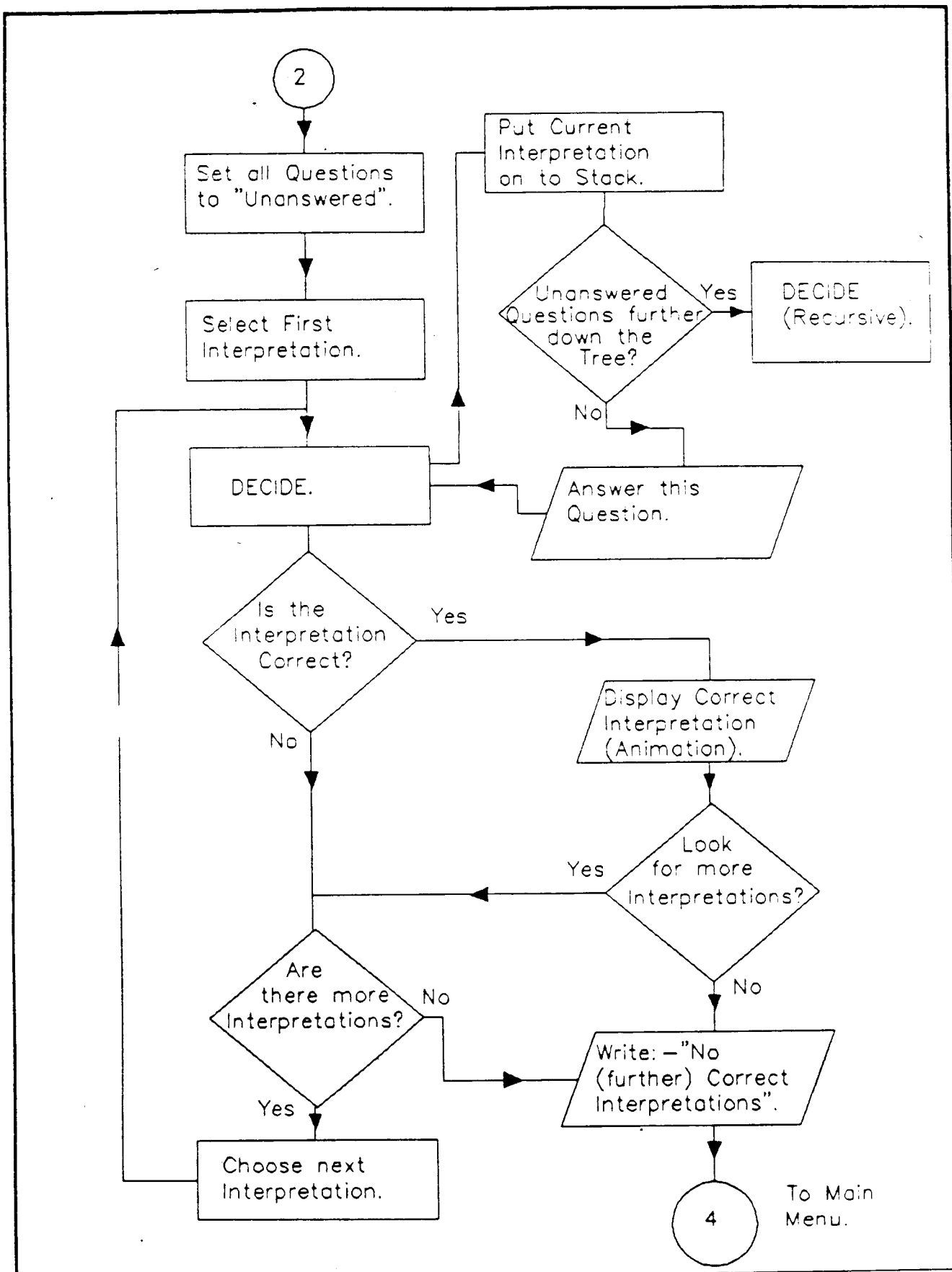

FIG. A1.4 Expert System Flowchart (Cont 2).

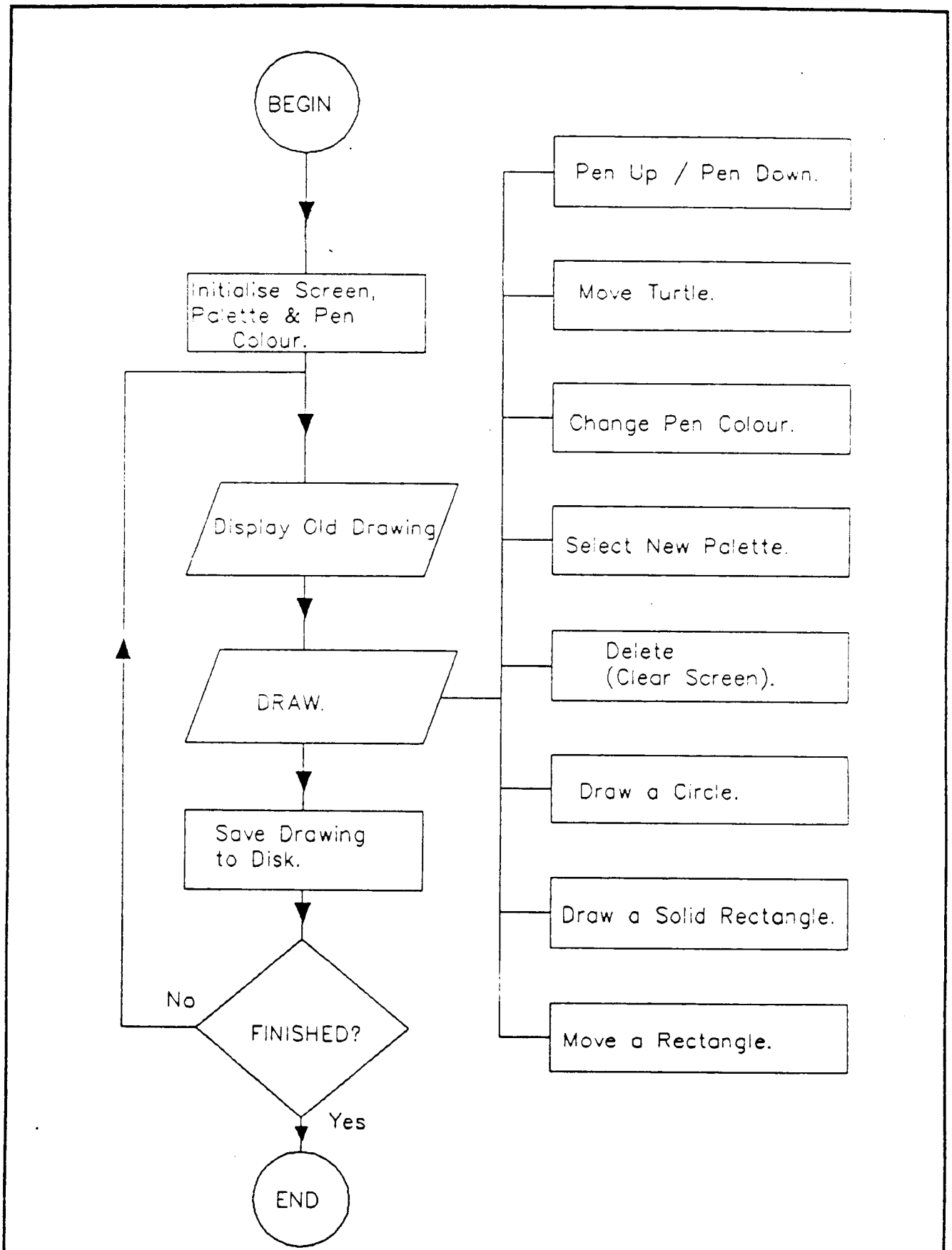

FIG. A1.5: Screen Painting Module Flowchart.

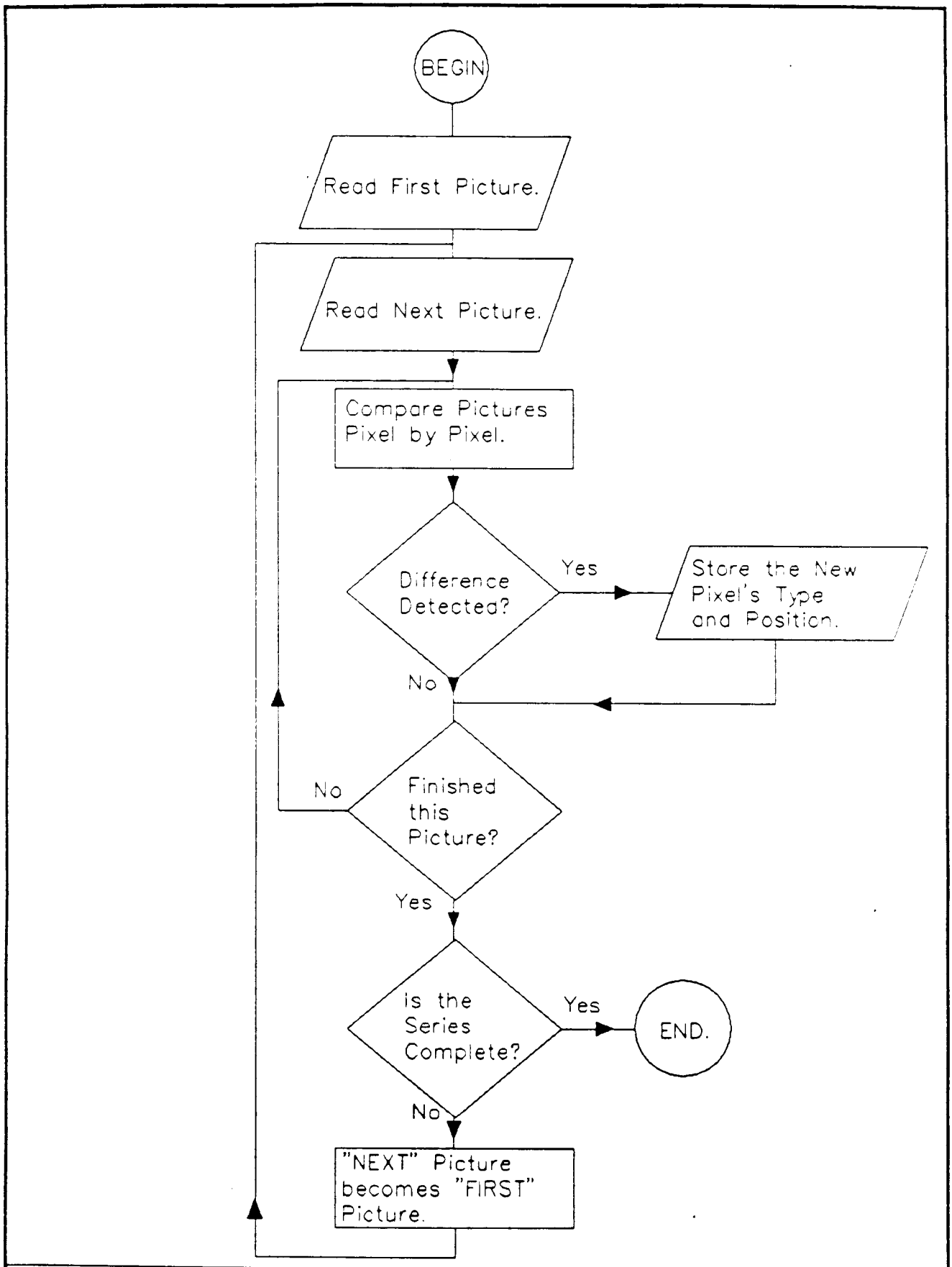

FIG. A1.6: Data Compression Module Flowchart

APPENDIX II

DOCUMENTATION OF PROGRAM SOURCE

Procedures and functions which are defined by Turbo Pascal 3.0 are not mentioned, while those defined in the Turbo Graphix Toolbox are listed. Full documentation for these routines can be found in the Turbo Graphix Toolbox user's manual. All other procedures and functions are listed in alphabetical order under the program in which they occur. The syntax of the routine is provided if the routine takes arguments. A complete listing of all modules is contained in Appendix II.

INPUT MODULE

Main Program

The Turbo Graphix Toolbox routines are initiated, and default settings selected via the set_default procedure. Data for the ECG are read and traced on the screen by the get_12lead_data procedure. The pop-up menu is called (Menu procedure) and returns a selection. The procedure appropriate to the selection is called. The pop-up menu call is repeated until the Quit option is selected.

Subroutines.

caliper1

Deletes all calipers from the displayed screen by copying the contents of a virtual screen represented in RAM into the actual screen. The set_calip procedure is called to draw the first caliper onto the new screen. The update_position procedure is called to determine the new position for the caliper, and the process of deleting all calipers and

redrawing the first caliper is repeated. This creates the illusion of the caliper moving across the screen.

caliper2

As for caliper1, but the first caliper is displayed and the second caliper moved.

caliper3

As for caliper1, but the first and second calipers are displayed and the third caliper moved.

caliper4

As for caliper1, but the first, second and third calipers are displayed and the fourth caliper moved.

cal_1_2

As for caliper1, but the first and second calipers are moved together.

cal_3_4

As for caliper1, but the first and second calipers are displayed and the third and fourth calipers moved together.

cal_all

As for caliper1, but all four calipers are moved together.

DefineHeader

see Turbo Graphix Toolbox user's manual.

definewindow

see Turbo Graphix Toolbox user's manual.

defineworld

see Turbo Graphix Toolbox user's manual.

display_12_lead

Restores the display of the twelve lead ECG to the screen.

display_measurements

Displays the identification of the P, QRS and T waves. Not implemented.

display_rhythm_strip

Restores the display of the rhythm strip to the screen. Not implemented.

do1line

Syntax : do1line(integer);

Do1line is called by scrdump to send the line of the screen specified by integer to the printer.

do2line

Syntax : do2line(integer);

As for do1line. The procedure is repeated to increase the vertical scaling as the screen resolution of 640 x 200 pixels would otherwise give a long, thin printout.

drawbox

Syntax: drawbox(height, length, line, column);

Drawbox draws the outline of a rectangle of the height and length specified. The top left corner of the box is in the line and column specified. As the procedure works with text characters it assumes a screen of 80 columns by 25 lines. It is used to outline the menu choices in the pop-up menu routines.

drawline

see Turbo Graphix Toolbox user's manual.

drawpoint

see Turbo Graphix Toolbox user's manual.

drawwave

Syntax : drawwave(array);

Drawwave plots 2.5 seconds of a single lead, digitised at 250 Hz, on the screen.

draw_background

Draws the template for the 12 lead ECG. i.e. draws twelve

boxes each with the appropriate heading.

getheadings

Defines the headings to be used by all the sub menus.

get_12lead_data

Reads data for eight of the twelve ECG leads from disk, and calculates the remaining four leads. The values are scaled to read in millivolts. The draw_background procedure is called, and the twelve lead ECG is plotted by calling the drawwave procedure.

get_rhythm_data

Similar to get_12_lead_data. Not implemented.

intro_message

Displays an introductory message. Not implemented.

Menu

Syntax : Menu(new choice, old choice);

Menu calls the main menu and sub menu procedures to set up the pop-up menu. It sets the pop-up menu to the choice previously selected. It returns the new choice as a two digit decimal integer. The first digit represents the choice from the main menu, the second digit that from the sub menu.

moveHor

see Turbo Graphix Toolbox user's manual.

moveVer

see Turbo Graphix Toolbox user's manual.

print_screen

Print_screen calls the scrdump procedure, then sets the choice from the menu back to what it was before the Print_screen procedure was called.

pulldown_mainmenu

Syntax : pulldown_mainmenu(old choice, new choice, show submenu, quit);

This procedure defines the choices available on the main menu, and calls the sub menu when required.

restorewindow

see Turbo Graphix Toolbox user's manual.

scrdump

Scrdump sends a scaled copy of the screen to the printer. It is specific for EPSON compatible printers, and is much faster than the standard PrtSc keyboard function.

selectscreen

see Turbo Graphix Toolbox user's manual.

selectwindow

see Turbo Graphix Toolbox user's manual.

selectworld

see Turbo Graphix Toolbox user's manual.

setbackground

see Turbo Graphix Toolbox user's manual.

setforegroundcolor

see Turbo Graphix Toolbox user's manual.

setlinestyle

see Turbo Graphix Toolbox user's manual.

setVstep

see Turbo Graphix Toolbox user's manual.

set_calip

Syntax : set_calip(position1, position2);

Draws the two calipers at the position specified. If the position specified is zero then the caliper is not drawn. The

procedure detects whether zoom or standard mode is selected and draws the appropriate calipers.

set_defaults

Initial values for variables are defined. The size and scaling of the windows used to draw the 12-lead ECG, for Zoom mode and for the pop-up menu system are defined. Pop-up menu variables are set, and all calipers are set to appear on the left of the 12-lead ECG trace.

storewindow

see Turbo Graphix Toolbox user's manual.

submenu

Syntax : submenu(submenu number, new choice, old submenu choice, quit);

Sets up the submenu indicated the submenu number, and returns the new choice selected.

unzoom

Adjusts the position of all the calipers for the twelve lead ECG, and calls the display_12_lead procedure to restore the display of the twelve lead ECG to the screen.

update_position

Syntax : update_position(position, tab, next key);

This procedure updates the position of a caliper. If the left or right arrow key is pressed continuously the position is changed in steps of ten, if pressed intermittently then the change occurs in steps of one. Each step consists of one pixel in zoom mode and four pixels in standard mode.

zoom2

Syntax : zoom2(cal1, cal2, cal3, cal4, lead number, lead title);

Displays the lead number selected on a full size screen, and

scales the caliper positions accordingly.

EXPERT SYSTEM MODULE

Main program

The program displays the title screen and calls the Initialise procedure. A knowledge base is compiled into memory via the ReadInput procedure, and checked for errors via the Audit procedure. The MainMenu procedure is then called. If errors have been detected in the knowledge base, the MainMenu call is repeated. This gives the user the opportunity to select a new knowledge base.

Subroutines.

abort

Syntax : abort(text message, integer);

This procedure displays the text message and integer provided, and sets flags to indicate that the knowledge base input has errors in it and cannot be used.

and_tree

The and_tree function is called by the or_tree function. If the AND symbol is present, the function accesses the next symbol via the NextSymbol procedure, and calls itself recursively. If the AND symbol is absent the function calls the not_tree function.

Animate

This procedure searches the logged drive for the first picture file in the series that corresponds to the decision that has just been validated. If this file is not found an error message is displayed and the procedure terminates. If the file is present, the expand procedure is called repeatedly until a key is pressed.

answerto

Syntax : answerto(question pointer, expected answer);

This function displays the question referred to by the question pointer on the screen, as well as all the available answers. If the user enters a digit corresponding to one of the available answers, the "question answered" flag is set and the function returns this choice. If the "W(hy)" option is selected, the text string on the top of the WHY STACK is displayed. Further "Why" requests will display the next text string on the WHY STACK. A choice of "0" sets the "Undo" flag and exits the function. A choice of "Hint" displays the "expected answer" provided. The "expected answer" is obtained from the decision tree of the current rule, and is the answer that will validate the decision currently pursued.

answer_rec

Syntax : answer_rec(answer pointer);

This function reads the possible answers to a question. The function calls the ReadStr function to read the text associated with the answer, and returns a pointer to this text.

Audit

The Audit procedure scans all the questions and decisions stored in RAM, and compiles a set of questions declared and decisions declared. The stored rules are then scanned, and a set of questions and a set of decisions referred to by the rules are compiled. If the sets of questions and decisions declared do not exactly match the sets of all questions and decisions referred to by the rules, an error results via the Audit_error procedure.

Audit_condition

This procedure is called by the Audit procedure to move through the decision tree of a particular rule.

Audit_error

This procedure is called if the Audit procedure detects an error in the stored knowledge base. The procedure determines the nature of the error, e.g. questions declared but not used, or questions used but not declared, and writes the appropriate error message.

Be_an_expert

This procedure controls the pseudo forward chaining search strategy. The reset_memory procedure is called as previous answers to questions are ignored. The first decision is selected, and the decide function is called to return a Boolean value which indicates whether the selected decision is valid (true) or not. If the "undo" flag has been set (in the answerto procedure, called by the satisfied function, which is called by the decide function) the reset_last procedure is called to "unanswer" the most recently answered question, and the decide function is called again. Once a decision has been found to be true, the procedure calls the animate procedure if requested to do so by the user. If the decision is valid the user can also request that the next decision be selected and the process repeated. If the decision is invalid, the next decision is automatically selected.

clearscreen

This procedure clears the screen and checks for I/O errors. If an error occurs, a message is displayed and the program terminates.

decide

Syntax : decide(decision pointer);

The decide function examines the status of the decision indicated by the decision pointer. If the status is undecided, the satisfied function is called and the status set to "pending". The satisfied function may call the decide function recursively if an intermediate decision is present. When the satisfied function terminates, the decision status is set to decided, and the decide function returns a Boolean value to indicate if the decision referred to is True or False. If the status is decided , the boolean value for the decision referred to is returned without further processing. If the status is "pending", this will result in an error as it indicates a circular rule definition.

decnnumber

This function calls the Expectnum function to ensure that the decision number is less than the maximum decision number permitted, and returns that number.

DTAtoDIR

This procedure is called from DynaDIR. It accesses the disc directory and returns a record containing the extended directory information on a disk file.

Dump

This procedure calls the printdecision, printquestion and printrule procedures to output a listing of the current knowledge base.

DynaDIR

Syntax : DynaDIR(Directory mask);

This function accesses a disc directory and returns a pointer

to a linked list of records of ASCII characters which contain the disk directory information. The procedure DTAtoDIR is called to access each directory entry.

expand

This procedure is called from the animate procedure. The expand procedure uncompresses the first picture file in a series and displays the picture on the screen. The next call to the expand procedure will display the next picture file. When no further picture files can be found on the logged drive, the series is repeated. The coding for this procedure in the listing provided in Appendix II assumes that picture files have been compressed according to the "String algorithm." Refer to the expand procedure in the Data Compression Module for the coding to uncompress files according to the Delta algorithm.

Expect

Syntax : expect(a symbol, error message);

If the latest symbol matches the "a symbol" provided, the NextSymbol procedure is called. If not, an error message results which is not ignored.

Expectnum

Syntax : Expectnum(whole number);

This function calls the Expect procedure to check that the latest symbol matches the "number" symbol. If there is a mismatch an error results which cannot be ignored. If no mismatch is found, and the number is less than the "whole number" provided, the function returns the number. If the number is not less than the "whole number", an error results which cannot be ignored.

factor

The factor function is called by the not_tree function. If the "left parenthesis" symbol is present, the function accesses the next symbol via the NextSymbol procedure, and the or_tree function . An error results if a "right parenthesis" symbol is not detected when control is returned from the or_tree function. If a "Question" or "Decision" symbol is detected, the question or decision number are entered in the decision tree. If other symbols are detected, an error results.

Ignore

Syntax : Ignore(a symbol);

If the latest symbol matches the "a symbol" provided, the NextSymbol procedure is called.

Initialise

This procedure marks the position on the RAM heap. All data stored on the heap above this mark belong to the current knowledge base. Flags are set to default values and pointers are set to nil.

InitialiseRead

This procedure asks the user to select the disk file containing the knowledge base. A list of all files with a .txt extension on the logged drive are displayed in a numbered list. The DynaDIR function is used to obtain this list. The user has the option of changing the logged drive and the list will be changed to reflect the files on the drive selected. Once a file is selected it is defined as the current knowledge base file, and the first line in the file is read.

List_Decisions

This procedure displays a list of all available decisions in the current knowledge base. It is called from the Be_an_expert procedure.

MainMenu

The procedure requests the user to choose one of five options. Selection of the backward chaining strategy ("Choose an interpretation from a list") will result in a call to the User_Validate procedure. The pseudo forward chaining option ("Get step by step guidance") will call the Be_an_expert procedure. A request to display the current knowledge base will call the dump procedure. A request to read in a new knowledge base will release (delete) all data stored on the RAM heap above the position marked in the Initialise procedure, and call the Initialise procedure and then the ReadInput procedure.

NextSymbol

This procedure converts the knowledge base language into "symbols" understood by the inference engine. It determines whether a character read from the knowledge base file is the beginning of a key word, a quote string or an integer, and calls the procedures ReadAKeyWord, ReadStr or Readnumber, respectively. Other legal characters, e.g. "&" signs or brackets, are interpreted. Unknown characters are noted in an error message, but ignored.

not_tree

The not_tree is called by the and_tree function. If the "NOT" symbol is present, the function accesses the next symbol via the NextSymbol procedure, and calls itself recursively. If the "NOT" symbol is absent the "factor" procedure is called.

or_tree

The or_tree function is called by the ReadRule procedure. If the "OR" symbol is present, the function accesses the next symbol via the NextSymbol procedure, and calls itself recursively. If the "OR" symbol is absent the function calls the "and_tree" function.

pop

Removes the top entry from the reason stack.

pop_decision

Syntax :pop_decision(whole number);

This function removes the top entry from the decision stack and returns the pointer to this entry.

pop_question

Syntax :pop_question(whole number);

This function removes the top entry from the question stack and returns the pointer to this entry.

press_any_key

Displays the message "Press any key to continue.", and waits for a key to be pressed.

press_one_of

Syntax : press_one_of(set of characters);

This function reads the keyboard until a character is entered which is in the set of characters provided. It returns this character.

printcond

Syntax : printcond(pointer);

This is a recursive procedure which is called by printrule to print out the rules stored in a decision tree.

printdecision

Syntax : printdecision(decision pointer);

If the pointer is not nil, the procedure prints the word "D :", followed by the line of text indicated by the decision pointer.

printquestion

Syntax : printquestion(question pointer);

If the pointer is not nil, the procedure prints the word "Q :", followed by a line of text for the question. It then calls the printwhy procedure to print out the reason for the question, and then prints out the possible answers to the question using the writelntext procedure.

printrule

Syntax : printrule(rule pointer);

If the pointer is not nil, the procedure prints the rule indicated by the pointer. The procedure calls the procedures printcond and printwhy.

printwhy

Syntax : printwhy(text pointer);

If the pointer is not nil, the procedure prints the word "WHY :", followed by the line of text indicated by the pointer.

push

Syntax : push(pointer);

Puts the pointer onto the reason stack, and checks for stack overflow.

push_decision

Syntax : push_decision(whole number);

Adds the decision corresponding to the whole number to the decison stack.

push_question

Syntax : push_question(whole number);

Adds the question corresponding to the whole number to the question stack.

quesnumber

This function calls the Expectnum function to ensure that the question number is less than the maximum question number permitted, and returns the question number if it is less.

ReadAKeyWord

This procedure is called by the NextSymbol procedure if the character variable (ch) contains the first letter of a knowledge base reserved word. The rest of the reserved word is read with the ReadCh procedure. If unexpected characters are encountered the error message "Unrecognised keyword" will result via the SayError procedure. Abbreviations are accepted, and do not result in an error.

ReadAnItem

This procedure examines the latest symbol, obtained from the NextSymbol procedure. If this symbol indicates the beginning of a question, a decision or a rule, the procedures ReadQuestion, ReadDecision or ReadRule are called. If the symbol is a full stop, the NextSymbol procedure is called. If any other symbol is detected (e.g. an answer) it is ignored, but results in an error message via the SayError procedure.

ReadCh

This procedure is part of the ReadInput procedure. It reads a line (eighty characters) of the file containing the knowledge base when called for the first time. A dot is placed on the screen as each line is read. It sets a

character variable to equal the first character on the line. If the first letter of the line is a small letter, it is converted to a capital letter. When called again it will set the character variable to the next character on the line. When all the characters have been allocated in this fashion, the next line will be read. If a character indicates that what follows is a comment, the rest of the line is ignored and the next line is read.

ReadDecision

This procedure is called from the ReadAnItem procedure. It calls the function decnnumber . If no error results the decision is read and the decision status set to "undecided". If the decision is associated with text, this is read and the decision classed as a final decision. If the text is absent the decision is classed as an intermediate decision. If a decision for the decision number has been read previously, this results in an error.

ReadInput

This procedure first calls the InitialiseRead procedure. The procedure ReadAnItem is called repeatedly until the symbol which indicates that the end of the knowledge base has been reached is encountered.

ReadInteger

Syntax : Readinteger(screen line, prompt message, number, maximum number);

ReadInteger displays the "prompt message" on the "screen line" provided. It reads the keyboard until a whole number is entered which is less than the "maximum number" provided. It returns this whole number.

ReadNumber

This procedure is called by the NextSymbol procedure when the character variable contains a digit character. The digit character is converted to an integer, and the ReadCh procedure called. If this also returns a digit character, both are converted to one integer. This is repeated until a non-digit character is detected, or until the integer is above an upper limit. The latter will result in an error message via the Sayerror procedure.

ReadQuestion

This procedure calls the ReadStrStr procedure to read the text associated with a question. The readwhy procedure is called to read the explanation for the question, and the answer_rec to return the possible answers to the question. The answers are assembled into a linked list.

ReadRule

This procedure calls the rulenumber function to ensure that the rule number does not exceed the maximum number of rules permitted. It then checks to ensure that the rule number has not been used before - defining two rules with the same number results in an error. The decision tree is assembled by calling the or_tree function, which in turn will call other functions if the "or" symbol is not present. The ReadRule procedure calls the SayError procedure if the rule does not begin with an "IF" symbol, and ends with a "THEN" symbol and a "Decision" symbol.

ReadStoStr

Syntax: ReadStoStr(text pointer, error message, severity);

ReadStoStr calls the procedure StoreText if the latest symbol indicates a text string. If the latest symbol indicates

anything else, an error results via the SayError procedure. If this is not severe it is ignored.

ReadStr

Syntax :ReadStr(quotation character);

This procedure is called by NextSymbol to read a string of characters enclosed in the quotes specified. The ReadCh procedure is used. If the string length exceeds a line an error message will result.

ReadWhy

Syntax : ReadWhy(text string);

If the latest symbol matches the "why" symbol, the NextSymbol procedure is called. The ReadStoStr procedure is called to read the text string. The ReadWhy returns this string as the "text string" variable.

Recover

This procedure calls the procedure NextSymbol repeatedly until a full stop or end of file token is detected.

Reset_last

This procedure sets the status of the question just answered to "unanswered", and sets the status of the current rule to "undecided." It is used to chain back to the previous question.

Reset_memory

This procedure sets all questions to "unanswered" and all decisions to "undecided".

rulenumber

This function calls the Expectnum function to ensure that the rule number is less than the maximum rule number permitted, and returns that number.

Satisfied

Syntax :Satisfied (decision tree node);

The satisfied function evaluates the decision tree, beginning at the decision tree node provided. If the node is a question, the answerto function is called. If the node is an intermediate decision, the decide procedure is called. All other nodes result in the satisfied function moving through the tree by calling itself recursively. The function returns the value "True" or "False".

SayError

Syntax : SayError(message, severity);

This procedure clears the screen and writes out the last line input. It indicates where an error has been detected in this line, and writes the error message to screen. If the error is not severe, it is ignored. If the error is severe, the user can instruct the procedure to continue reading in the knowledge base. In this case the Recover procedure is called. If the user indicates that the procedure must not continue, the program is terminated.

storetext

Syntax : storetext(string, pointer);

The string of characters is stored on the heap, and the pointer is set to point to the string.

User_Validate

This procedure controls the backward-chaining search strategy. The List_decisions procedure is called, and the user is requested to select an option from the list of possible decisions. If previous answers to questions are to be ignored, the reset_memory procedure is called. The decide function is called to return a boolean value which

indicates whether the selected decision is valid (true) or not. If the "undo" flag has been set (in the "answerto" procedure, called by the satisfied function, which is called by the decide function) the Reset_last procedure is called to "unanswer" the most recently answered question, and the decide function is called again. Once a decision has been found to be true, the procedure calls the animate procedure if requested to do so by the user.

Worldmap

This procedure displays the title screen, which consists of a map of the world with the title message across it.

writelndecn

Syntax : writelndecn(decision pointer);

If the decision indicated is not an intermediate decision, write the comment associated with that decision, else write empty parentheses. The screen cursor is then set to the beginning of the next line.

writelntext

Syntax : writelntext(pointer);

If the pointer does not equal nil, the string that the pointer points to is written to the screen. The screen cursor is set to the beginning of the next line.

writetext

As for writelntext, but the screen cursor is not set.

ANIMATOR - SCREEN PAINTING MODULE

Main program.

The program calls for the name of a picture series. The first picture from the series is placed on the screen if a legal name is provided. The drawscreen procedure is then called. When this procedure terminates, the pictures created are displayed sequentially from beginning to end and back, until a key is pressed.

Subroutines.

copywindow

This procedure allows the user to select the diagonally opposite corner of a rectangle once the first corner has been selected. The drawing contained in this rectangle is then copied to another position on the screen, once the top left corner of the new position has been defined.

deletewindow

This procedure allows the user to select the diagonally opposite corner of a rectangle once the first corner has been selected, and then to fill the rectangle with a selected solid colour. This deletes the drawing under the rectangle.

drawcircle

This procedure allows the user to select the radius of a circle once the centre has been selected, and then describes the circumference of the circle.

drawscreen

This procedure reads the keyboard, and calls the selected procedure. Most of these procedures are standard procedures defined by Turbo Pascal's extended graphics capabilities, and some have been specially written. Facilities for saving a

sequential series of pictures to disk are provided.

initialise

This procedure sets defaults for the colour and palette selected, and positions the turtle in the middle of the screen.

ANIMATOR - DATA COMPRESSION MODULE

Main program.

The program calls for the name of a series of picture files. The first picture in the series is assumed to have the extension "a". The compress and expand procedures are called. The program then terminates.

Subroutines.

compress

The first picture file in the picture series is read, and stored in a buffer. The second file is read into a second buffer. The two buffers are compared byte by byte, and the position and type of any bytes that differ noted in a data buffer. The data buffer is stored as the compressed version of the second picture file. The process is then repeated for the rest of the picture series. The blockread and blockwrite commands are used to increase the speed of data transfer between memory and disk.

expand

The first, uncompressed file in the picture series is read and the picture displayed on the screen. The first compressed file (i.e. the second picture) is read. The picture is recreated and displayed on the screen. The process is repeated until no further compressed files are found in the

series. When this happens the first picture is displayed again, and the entire series repeated. This continues until a key is pressed.

APPENDIX 3

PROGRAM LISTINGS

APPENDIX III

PROGRAM LISTINGS

INPUT MODULE

```
PROGRAM ecg;

{————————————————————————————}
{  These include files contain the standard routines      }
{  of the TURBO GRAPHIX TOOLBOX.                          }
{————————————————————————————}
{$I a:typedef.sys}
{$I a:graphix.sys}
{$I a:kernel.sys}
{$I a:windows.sys}

{————————————————————————————}
{            Data structures                              }
{————————————————————————————}

TYPE array625        = ARRAY[1..625] OF REAL;
 string20          = STRING[20];

VAR data             : ARRAY[1..12] OF array625;
 option,pull_choice,old_choice,current_screen,
 c1,c2,c3,c4,menu_i_Glb,zoom_offset: INTEGER;
 sub_on,move_rt,move_Lt,twelve_lead,rhythm_strip:BOOLEAN;
 key              : CHAR;
 line_no          : INTEGER;

{————————————————————————————}

PROCEDURE set_defaults;

BEGIN
 c1:=10;
 c2:=10;
 c3:=10;
 c4:=10;
 menu_i_Glb:=0;
 move_rt:=FALSE;
 move_lt:=FALSE;
 sub_on:=FALSE;
 pull_choice:=11;
 old_choice:=11;
 key:='[';
 setlinestyle(0);
 definewindow(1,1,7,11,16);      {used in main menu }
 definewindow(4,0,0,19,57);      {used in draw_background}
 definewindow(7,0,10,79,57);     {used to draw ecgs}
 definewindow(5,0,10+62,79,57+62); {     "     }
 definewindow(6,0,10+62+62,79,57+62+62); {     "     }
 defineworld(6,0,-2000,2500,2000);
 definewindow(8,0,0,xmaxglb,ymaxglb-16); {12 lead ecg}
 defineworld(8,0,0,2500,ymaxglb-16);
 definewindow(10,5,5,xMaxGlb-5,YmaxGlb-5);{used in zoom
procedures}
 defineworld(10,0,-2000,625,2000);
 definewindow(11,3,20,75,160);
END;

{————————————————————————————}
{    This procedure draws a rectangle in text mode        }
{————————————————————————————}

PROCEDURE
drawbox(h_length,v_length,h_offset,v_offset:INTEGER);

 VAR i,j:INTEGER;

BEGIN
 IF h_offset0 THEN gotoxy(h_offset+1,v_offset);
 write(chr(213));
 FOR i:= 1 TO H_length DO write(chr(205));
 writeln(chr(184));
 IF h_offset0 THEN gotoxy(h_offset+1,v_offset+1);
 FOR j:= 1 TO v_length DO
  BEGIN
   write(chr(179));
   FOR i:=1 TO H_length DO write(' ');
   writeln(chr(179));
   IF h_offset0 THEN gotoxy(h_offset+1,v_offset+1+j);
  END;
 write(chr(212));
 FOR i:= 1 TO H_length DO write(chr(205));
 writeln(chr(190));
END;

{————————————————————————————}
{ this procedure is a modification of the "hardcopy" procedure
}
{ found in the turbo graphix toolbox. It prints four pixels    }
{ per printer line i.e. scales the Y=axis up by two.          }
{————————————————————————————}

OVERLAY PROCEDURE
scrdump(inverse:BOOLEAN;mode:byte); { EPSON }
 VAR i,j,top:INTEGER;
  ColorLoc,PrintByte:byte;

 PROCEDURE do1line(top:INTEGER);
  FUNCTION ConstructByte(j,i:INTEGER):byte;
   CONST Bits:ARRAY [0..7] OF byte = (128,64,32,16,8,4,2,1);
   VAR CByte,k:byte;
  BEGIN
   i:=i SHL 3;
   CByte:=0;
   FOR k:=0 TO TRUNC(top DIV 2) DO
    IF PD(j,i+k) THEN BEGIN
      CByte:=CByte OR Bits[2*k];
      CByte:=CByte OR Bits[2*k+1];
     END;
```

```
    ConstructByte: = CByte;
   END;
  BEGIN
   IF mode = 1 THEN write(lst, ^['L')
   ELSE write(lst, ^['*',chr(mode));

  write(lst,chr(lo(XScreenMaxGlb + 1)),chr(Hi(XScreenMaxGlb + 1))
  );
   FOR j: = 0 TO XScreenMaxGlb DO
    BEGIN
     PrintByte: = ConstructByte(j,i);
     IF inverse THEN PrintByte: = NOT PrintByte;
     write(lst,chr(PrintByte));
    END;
   IF mode4 THEN writeln(lst);
  END;

  PROCEDURE do2line(top:INTEGER);
   FUNCTION ConstructByte(j,i:INTEGER):byte;
    CONST Bits:ARRAY [0..7] OF byte = (128,64,32,16,8,4,2,1);
    VAR CByte,k:byte;
   BEGIN
    i: = i SHL 3;
    CByte: = 0;
    FOR k: = 0 TO TRUNC(top DIV 2) DO
     IF PD(j,i + k + 4) THEN  BEGIN
       CByte: = CByte OR Bits[2*k];
       CByte: = CByte OR Bits[2*k + 1];
      END;
    ConstructByte: = CByte;
   END;
  BEGIN
   IF mode = 1 THEN write(lst, ^['L')
   ELSE write(lst, ^['*',chr(mode));

  write(lst,chr(lo(XScreenMaxGlb + 1)),chr(Hi(XScreenMaxGlb + 1))
  );
   FOR j: = 0 TO XScreenMaxGlb DO
    BEGIN
     PrintByte: = ConstructByte(j,i);
     IF inverse THEN PrintByte: = NOT PrintByte;
     write(lst,chr(PrintByte));
    END;
   IF mode4 THEN writeln(lst);
  END;

 BEGIN
  top: = 7;
  ColorLoc: = ColorGlb;
  ColorGlb: = 255;
  mode: = mode AND 7;
  IF (mode = 5) OR (mode = 0) THEN mode: = 4;
  write(lst, ^['3'#24);
  FOR i: = 0 TO ((YMaxGlb + 1) SHR 3)-1 DO BEGIN
 do1line(7);do2line(7);END;
  i: = ((YMaxGlb + 1) SHR 3);
  IF (YMaxGlb + 1) AND 70 THEN
  BEGIN do1line((YMaxGlb + 1) AND 7);do2line((YMaxGlb + 1)
 AND 7);END;
  writeln(lst, ^['2');
  ColorGlb: = ColorLoc;
 END;

{-------------------------------------}
PROCEDURE print_screen;

BEGIN
 scrdump(FALSE,5);
 pull_choice: = old_choice;
END;


{-------------------------------------}
{ This procedure contains the pop-up (or pulldown, if you      }
{ prefer) menu system.                          }
{-------------------------------------}
OVERLAY PROCEDURE Menu(VAR
oldchoice,finalchoice:INTEGER);


 CONST maxlength = 12;

 TYPE
  stringArray = ARRAY[1..maxlength] OF STRING[15];

 VAR quit:BOOLEAN;
  newchoice,subchoice,mainmenu_choice:INTEGER;

{===============================
==============}

 PROCEDURE pulldown_mainmenu(VAR
choice,newchoice:INTEGER;VAR subon,mquit:BOOLEAN);

  VAR str:ARRAY[1 .. 7] OF STRING[11];
   I:INTEGER;
   ch,ch2:CHAR;

 BEGIN
  str[1]: = 'calipers  ';
  str[2]: = ' unzoom   ';
  str[3]: = 'zoom limb ';
  str[4]: = ' zoom V   ';
  str[5]: = 'new screen';
  str[6]: = '  quit    ';
  str[7]: = '  HELP    ';
  gotoxy(1,1);
  drawbox(78,1,0,0);
  gotoxy(2,2);write(str[1],str[2],str[3],str[4],str[5],str[6],str[7]);
  copyscreen;
  I: = menu_I_Glb;
  selectwindow(1);
  invertwindow;
  IF move_rt THEN BEGIN
    I: = I + 11;
    IF i76 THEN BEGIN
      I: = I-77;
      movehor(-66,TRUE);
     END
```

```
    ELSE movehor(11,TRUE);
    selectscreen(2);
    copyscreen;
    selectscreen(1);
    invertwindow;
    read(kbd,ch);
   END
 ELSE IF move_lt THEN BEGIN
    i:=i-11;
    IF i THEN BEGIN
      i:=i+77;
      movehor(66,TRUE);
     END
    ELSE movehor(-11,TRUE);
    selectscreen(2);
    copyscreen;
    selectscreen(1);
    invertwindow;
    read(kbd,ch);
   END;
 move_rt:=FALSE;
 move_lt:=FALSE;
 IF NOT subon THEN REPEAT
    IF (ch = '') THEN
     BEGIN
      read(kbd,ch2);
      IF (ch2='M')
       THEN BEGIN
        i:=i+11;
        IF i76 THEN BEGIN
          i:=i-77;
          movehor(-66,TRUE);
         END
        ELSE movehor(11,TRUE);
        selectscreen(2);
        copyscreen;
        selectscreen(1);
        invertwindow;
        read(kbd,ch);
       END
      ELSE IF ch2=('K')
       THEN BEGIN
        i:=i-11;
        IF i THEN BEGIN
          i:=i+77;
          movehor(66,TRUE);
         END
        ELSE movehor(-11,TRUE);
        selectscreen(2);
        copyscreen;
        selectscreen(1);
        invertwindow;
        read(kbd,ch);
       END
      ELSE IF (ch2='P') THEN ch:=chr(13);

    END
    ELSE read(kbd,ch);
    IF ch='p' THEN print_screen;
   UNTIL (ch '')
  ELSE ch:=chr(13);
  mquit:=TRUE;
  move_rt:=FALSE;
  move_lt:=FALSE;
  menu_l_Glb:=i;
  choice:=i DIV 11+1;
  newchoice:=choice;
  IF ch=(chr(13)) THEN BEGIN
    sub_on:=TRUE;
    mquit:=FALSE;
   END;
 END;

{————————————————————————————}
 PROCEDURE getheadings(menu_num:INTEGER;VAR
str:stringarray);

  VAR i: INTEGER;

 BEGIN
  FOR i:=1 TO maxlength DO
   str[i]:=' ';
  CASE menu_num OF
  1:BEGIN
    str[1]:=' one  ';
    str[2]:=' two  ';
    str[3]:=' three ';
    str[4]:=' four ';
    str[5]:=' one and two ';
    str[6]:=' three and four';
    str[7]:=' all  ';
   END;
  2:BEGIN
    str[1]:='to 12 lead ecg';
    str[2]:='to rhythm strip';
   END;
  3:BEGIN
    str[1]:='Std lead I ';
    str[2]:='Std lead II ';
    str[3]:='Std lead III';
    str[4]:='aVl';
    str[5]:='aVr';
    str[6]:='aVf';
   END;
  4:BEGIN
    str[1]:=' V1  ';
    str[2]:=' V2';
    str[3]:=' V3 ';
    str[4]:=' V4  ';
    str[5]:=' V5 ';
    str[6]:=' V6';
   END;
  5:BEGIN
    str[1]:=' 12 lead' ;
    str[2]:=' Rhythm strip ';
```

```
    str[3]:= ' Diagnose ';
   END;
  6:BEGIN
    str[1]:= ' yes   ';
    str[2]:= ' no ';
   END;
  7:BEGIN
    str[1]:= 'not used  ';
    str[3]:= 'not used  ';
    str[4]:= 'not used  ';
    str[5]:= 'not used  ';
    str[6]:= 'not used  ';
   END;
  END; {case}
 END;

{-------------------------------------}
 PROCEDURE submenu(VAR
num,new_choice,sub_choice:INTEGER;
  VAR sub_quit:BOOLEAN);

  VAR strArray:stringarray;
   i,x1,x2,y1,y2,menu_length,width:INTEGER;
   ch,ch2:CHAR;
   first_time:BOOLEAN;

{the following variables are used:-
              num - number of this submenu.
              menu_length-number of options in the menu.
              width - length of longest string.}


 BEGIN
  width:= 1;
  menu_length:= 0;
  setVstep(8);
  getheadings(num,strArray);
  REPEAT
   menu_length:= menu_length + 1;
  UNTIL strArray[menu_length]= ' ';
  menu_length:= menu_length-1;
  FOR i:= 1 TO menu_length DO
   IF width gth(strArray[i]) THEN width:= length(strArray[i]);
  writeln;
  x1:= (num*11-10);
  x2:= (x1 + width + 2);
 definewindow(2,x1,(whereY),x2,(whereY + (menu_length + 3)*9))
 ;
  storewindow(2);
  drawbox(width,menu_length,num*11-10,3);
  FOR i:= 1 TO menu_length DO
   BEGIN
    gotoxy(num*11-8,i + 3);
    write(strArray[i]);
   END;
  copyscreen;
  i:=0;
definewindow(3,(num-1)*11 + 2,8*(2),(num-1)*11 + width + 1,8*(3
));
  selectwindow(3);
  first_time:= TRUE;
  ch:= #27;
  REPEAT
   IF (ch= '')
    THEN
    BEGIN
     IF first_time THEN BEGIN
       first_time:= FALSE;
       ch2:= 'P'
      END
     ELSE read(kbd,ch2);
     IF (ch2= 'P')
      THEN BEGIN
       i:= i + 8;
       IF i8*menu_length THEN BEGIN
         i:= i-8*menu_length;
         moveVer(-8*(menu_length-1),TRUE);
        END
       ELSE moveVer(8,TRUE);
       selectscreen(2);
       copyscreen;
       selectscreen(1);
       invertwindow;
       read(kbd,ch);
      END
     ELSE IF ch2= ('H')
      THEN BEGIN
       i:= i-8;
       IF i0 THEN BEGIN
         i:= i + 8*menu_length;
         moveVer(8*(menu_length-1),TRUE);
        END
       ELSE moveVer(-8,TRUE);
       selectscreen(2);
       copyscreen;
       selectscreen(1);
       invertwindow;
       read(kbd,ch);
      END
     ELSE IF ch2= 'K' THEN BEGIN
       ch:= chr(26);
       move_Lt:= TRUE;
      END
     ELSE IF ch2= 'M'
      THEN BEGIN
       ch:= chr(26);
       move_Rt:= TRUE;
      END;
    END
   ELSE read(kbd,ch);
   IF ch= 'p' THEN print_screen;
  UNTIL (ch '');
  sub_quit:= FALSE;
  sub_choice:= 0;
```

```
 IF ch = (chr(13)) THEN BEGIN
    sub_choice: = (i DIV 8);
    sub_quit: = TRUE;
   END
 ELSE sub_on: = FALSE;
 restorewindow(2,0,0);
END;

{-------------------------------------}

BEGIN
 oldchoice: = finalchoice;
 mainmenu_choice: = oldchoice DIV 10;
 subchoice: = oldchoice MOD 10;
 newchoice: = mainmenu_choice,
 quit: = TRUE;
 REPEAT
pulldown_mainmenu(mainmenu_choice,newchoice,sub_On,quit
);
   IF (sub_On)
    THEN
submenu(mainmenu_choice,newchoice,subchoice,quit);
 UNTIL quit;
 finalchoice: = mainmenu_choice*10 + subchoice;
END;


{-------------------------------------}
{ This procedure plots the tracing: pairs of readings are }
{ plotted as a single point to increase speed.            }
{-------------------------------------}

OVERLAY PROCEDURE drawwave(wave:array625);

 VAR n:INTEGER;
  a,b:REAL;

BEGIN
 FOR n: = 1 TO 311 DO
  BEGIN
   a: = (wave[2*n] + wave[2*n-1])/2.0;
   b: = (wave[2*n + 2] + wave[2*n + 1])/2.0;
   IF abs(a-b) THEN BEGIN
     drawpoint(2*n,(a + b)/2);
    END
   ELSE drawline(2*n,a,(2*n + 2),b);
  END;
END;


{-------------------------------------}
{ Procedures to manage the caliper system.  }
{-------------------------------------}

PROCEDURE update_position(VAR
pos:INTEGER;tab:INTEGER;
 VAR next_key:CHAR);

CONST esc = #27;

VAR key:CHAR;
 increase:INTEGER;

BEGIN
 IF current_screen = 10 THEN increase: = 1
 ELSE increase: = 4;
 read(kbd,key);
 IF key = 'p' THEN print_screen;
 next_key: = key;
 IF key IN ['',chr(27)] THEN
  CASE key OF
  '' : BEGIN
    pos: = pos-abs(tab);
    IF pos THEN pos: = pos + 2500;
   END;
  '' : BEGIN
    pos: = pos + abs(tab);
    IF pos2500 THEN pos: = pos-2500
   END;
  esc : BEGIN
    IF NOT keypressed THEN
     next_key: = 'q'
    ELSE BEGIN
      read(kbd,key);
      IF keypressed THEN increase: = 10*increase;
      IF (key = 'M')
       THEN BEGIN
        pos: = pos + increase;
        IF pos2500 THEN pos: = pos-2500;
       END
      ELSE IF key = ('K')
       THEN BEGIN
        pos: = pos-increase;
        IF pos THEN pos: = pos + 2500;
       END;
     END;
   END;
  END; {case}
END;

{-------------------------------------}
PROCEDURE set_calip(k1:INTEGER;k2:INTEGER);

BEGIN
 IF current_screen = 10 THEN
  BEGIN
   IF (k10) AND (k1q) THEN drawline(k1,-1900,k1,1900);
   IF (k20) AND (k2q) THEN drawline(k2,-1900,k2,1900);
  END
 ELSE
  BEGIN
   drawline(k1,6,k1,46);
   drawline(k1,68,k1,108);
   drawline(k1,130,k1,170);
   IF k20 THEN BEGIN
```

```
      drawline(k2,6,k2,46);
      drawline(k2,68,k2,108);
      drawline(k2,130,k2,170);
    END;
  END;
END;

{-------------------------------------}
OVERLAY PROCEDURE caliper1(VAR k1:INTEGER;VAR
new_key:CHAR);

 CONST zero = 0;

BEGIN
 restorewindow(current_screen,0,0);
 selectworld(current_screen);
 selectwindow(current_screen);
 setlinestyle(85);
 IF new_key<>'q' THEN BEGIN
    gotoxy(1,24);write(' ':80);{clear previous commentline}
  END;
 copyscreen;
 REPEAT
  selectscreen(2);
  copyscreen;
  selectscreen(1);
  set_calip(k1,zero);
  update_position(k1,10,new_key);
 UNTIL new_key = 'q';
 setlinestyle(0);
END;

{-------------------------------------}
OVERLAY PROCEDURE caliper2(VAR k1,k2:INTEGER;VAR
new_key:CHAR);

 VAR a,b:INTEGER;

BEGIN
 restorewindow(current_screen,0,0);
 selectworld(current_screen);
 selectwindow(current_screen);
 setlinestyle(85);
 restorewindow(current_screen,0,0);
 set_calip(k1,0);
 copyscreen;
 WHILE (new_key<>'q')
  DO BEGIN
   selectscreen(2);
   copyscreen;
   selectscreen(1);
   set_calip(k2,0);
   a := round(abs(k1-k2)*4);
   IF k1-k2<0 THEN b:= round(abs(15000/(k1-k2)))
   ELSE b:= 0;
   gotoxy(10,24);write(a,' msec : ',b,' per minute ');
   update_position(k2,10,new_key);
  END;
 setlinestyle(0);
END;

{-------------------------------------}
OVERLAY PROCEDURE cal_1_2(VAR k1,k2:INTEGER;VAR
new_key:CHAR);

 VAR a,b,delta:INTEGER;

BEGIN
 restorewindow(current_screen,0,0);
 selectworld(current_screen);
 selectwindow(current_screen);
 setlinestyle(85);
 restorewindow(current_screen,0,0);
 copyscreen;
 delta := (k2-k1);
 a := round(abs(delta)*4);
 IF delta<0 THEN b := round(abs(15000/(delta)))
 ELSE b := 0;
 gotoxy(10,24);write(a,' msec : ',b,' per minute ');
 WHILE (new_key<>'q')
  DO BEGIN
   selectscreen(2);
   copyscreen;
   selectscreen(1);
   set_calip(k1,k1+delta);
   update_position(k1,delta,new_key);
  END;
 k2 := k1+delta;
 setlinestyle(0);
END;

{-------------------------------------}
OVERLAY PROCEDURE caliper3(VAR k1,k2,k3:INTEGER;VAR
new_key:CHAR);

 VAR a,b:INTEGER;

BEGIN
 restorewindow(current_screen,0,0);
 selectworld(current_screen);
 selectwindow(current_screen);
 setlinestyle(85);
 set_calip(k1,k2);
 setlinestyle(4);
 gotoxy(50,24);write(' ':30);
 copyscreen;
 a := round(abs(k1-k2)*4);
 IF k1-k2<0 THEN b:= round(abs(15000/(k1-k2)))
 ELSE b:= 0;
 gotoxy(10,24);write(a,' msec : ',b,' per minute ');
 setlinestyle(4);
 WHILE (new_key<>'q')
  DO BEGIN
   selectscreen(2);
   copyscreen;
   selectscreen(1);
```

```
    set_calip(k3,0);
    update_position(k3,10,new_key);
   END;
  setlinestyle(0);
END;

{-------------------------------------}
OVERLAY PROCEDURE caliper4(VAR
k1,k2,k3,k4:INTEGER;VAR new_key:CHAR);

 VAR a,b:INTEGER;

BEGIN
 restorewindow(current_screen,0,0);
 selectworld(current_screen);
 selectwindow(current_screen);
 setlinestyle(85);
 restorewindow(current_screen,0,0);
 set_calip(k1,k2);
 setlinestyle(4);
 set_calip(k3,0);
 a:=round(abs(k1-k2)*4);
 IF k1-k20 THEN b:=round(abs(15000/(k1-k2)))
 ELSE b:=0;
 gotoxy(10,24);write(a,' msec : ',b,' per minute ');
 copyscreen;
 WHILE (new_key'q')
  DO BEGIN
    selectscreen(2);
    copyscreen;
    selectscreen(1);
    set_calip(k4,0);
    a:=round(abs(k3-k4)*4);
    IF k3-k40 THEN b:=round(abs(15000/(k3-k4)))
    ELSE b:=0;
    gotoxy(50,24);write(a,' msec : ',b,' per minute ');
    update_position(k4,10,new_key);
   END;
  setlinestyle(0);
END;

{-------------------------------------}
OVERLAY PROCEDURE cal_3_4(VAR
k1,k2,k3,k4:INTEGER;VAR new_key:CHAR);

 VAR a,b,delta:INTEGER;

BEGIN
 restorewindow(current_screen,0,0);
 selectworld(current_screen);
 selectwindow(current_screen);
 setlinestyle(85);
 set_calip(k1,k2);
 setlinestyle(4);
 copyscreen;
 a:=round(abs(k1-k2)*4);
 IF k1-k20 THEN b:=round(abs(15000/(k1-k2)))
 ELSE b:=0;
 gotoxy(10,24);write(a,' msec : ',b,' per minute ');
 setlinestyle(4);
 delta:=k4-k3;
 WHILE (new_key'q')
  DO BEGIN
   selectscreen(2);
   copyscreen;
   selectscreen(1);
   set_calip(k3,k3+delta);
   a:=abs(delta*4);
   IF delta0 THEN b:=round(abs(15000/(delta)))
   ELSE b:=0;
   gotoxy(50,24);write(a,' msec : ',b,' per minute ');
   update_position(k3,delta,new_key);
  END;
 k4:=k3+delta;
 setlinestyle(0);
END;

{-------------------------------------}

OVERLAY PROCEDURE cal_all(VAR k1,k2,k3,k4:INTEGER;VAR
new_key:CHAR);

 VAR a,b,delta1,delta2,delta3:INTEGER;

BEGIN
 selectworld(current_screen);
 selectwindow(current_screen);
 restorewindow(current_screen,0,0);
 copyscreen;
 delta1:=k2-k1;
 delta2:=k4-k3;
 delta3:=k3-k1;
 a:=abs(delta1*4);
 IF delta10 THEN b:=round(abs(15000/(delta1)))
 ELSE b:=0;
 gotoxy(10,24);write((delta1*4),' msec :',b,' per minute ');
 a:=abs(delta2*4);
 IF delta20 THEN b:=round(abs(15000/(k1-k2)))
 ELSE b:=0;
 gotoxy(50,24);write((delta2*4),' msec : ',b,' per minute ');

 WHILE (new_key'q')
  DO BEGIN
    selectscreen(2);
    copyscreen;
    selectscreen(1);
    setlinestyle(85);
    set_calip(k1,k1+delta1);
    setlinestyle(4);
    set_calip(k3,k3+delta2);
    setlinestyle(85);
    update_position(k1,delta1,new_key);
    k3:=k1+delta3;
   END;
  k2:=k1+delta1;
  k4:=k3+delta2;
```

```
END;


{-------------------------------------}
PROCEDURE intro_message;

BEGIN
END;

{-------------------------------------}
{ Another procedure to plot the trace. It averages two points, }
{ and draws a line between the is they are far apart, or a     }
{ point between then if they are close.                        }
{-------------------------------------}
PROCEDURE draw_wave(num,n,offset:INTEGER);

 VAR
  a,b:INTEGER;

BEGIN
 a:=round((data[num,2*n]+data[num,2*n-1])/2.0);
 b:=round((data[num,2*n+2]+data[num,2*n+1])/2.0);
 IF abs(a-b) THEN BEGIN
    drawpoint(2*n+offset,(a+b)/2);
  END
 ELSE drawline(2*n+offset,a,(2*n+2)+offset,b);
END;



{-------------------------------------}
{ This procedure draws the template for the 12 lead ecg trace. }
{-------------------------------------}
PROCEDURE draw_background;

 VAR title:STRING[8];
  i,j,x_offset,y_offset:INTEGER;
BEGIN
 clearscreen;
 setheaderOn;
 setHeadertoTop;
 definewindow(4,0,0,19,57);
 storewindow(4);
 FOR i:= 0 TO 3 DO
  FOR j:= 0 TO 2 DO
   BEGIN
    x_offset:=i*20;
    y_offset:=j*62;
    restorewindow(4,x_offset,y_offset);
    selectworld(4);
    selectwindow(4);
    CASE (i*3+j+1) OF
    1:title:='Std I';
    2:title:='Std II';
    3:title:='Std III';
    4:title:='aVR';
    5:title:='aVL';
    6:title:='aVF';
    7:title:='V1';
    8:title:='V2';
    9:title:='V3';
    10:title:='V4';
    11:title:='V5';
    12:title:='V6';
    END {case};
    DefineHeader(4,title);
    drawborder;
   END;
END;

{-------------------------------------}
{ This procedure reads the data for 8 leads, calculates the    }
{ remaining four, and then constructs the display of all 12    }
{ leads                                                        }
{-------------------------------------}

PROCEDURE get_12lead_data;

 VAR OK:BOOLEAN;
  i,j,k,offset1,offset2:INTEGER;
  datum:INTEGER;
  filename,newfile:STRING[8];
  datafile:FILE OF INTEGER;

BEGIN
 OK:=TRUE;
 i:=0;
 j:=0;
 drawbox(46,1,9,10);
 REPEAT
  IF i=0 THEN
   BEGIN
    gotoxy(11,11);
    write(' enter name of 12 lead ecg... ');
    gotoxy(40,11);
    read(filename);
   END;
  i:=i+1;
  IF j=0 THEN j:=2              {read std leads II and III,}
  ELSE IF j=2 THEN j:=3         {then v leads into data[6..12]}
  ELSE IF j=3 THEN j:=7
  ELSE j:=j+1;
  newfile:=filename+chr(ord('0')+i);
  assign(datafile,newfile);
   {$I-} reset(datafile); {$I+}
  OK:=(IOresult=0);
  IF NOT OK THEN BEGIN
    i:=0;
    j:=0;
    gotoxy(11,11);
    write('cannot find this ecg..... ',filename,'   ');
    delay(3000);
   END
  ELSE BEGIN
```

```
    gotoxy(11,11);
    write('   WAIT                    ');
    gotoxy(19,11);FOR k: = 1 TO i DO write('.');
    FOR k: = 1 TO 625 DO
     BEGIN
      read(datafile,datum);
      data[j,k]. = (datum-2048)/0.205;
     END;
    close(datafile);
   END;
UNTIL ((OK) AND (i = 8)) OR (filename = 'stop');
BEGIN
 FOR k: = 1 TO 625 DO
  BEGIN
   IF (k MOD 105) = 0 THEN write('.');
     {std!}      data[1,k]: = data[2,k]-data[3,k];
     {a.r.t.
data[4,k]} = round((data[2,k] + data[3,k])/3.0-data[2,k]);
     {avl}       data[5,k]: = round((data[2,k]-2*data[3,k])/3.0);
     {avf}       data[6,k]: = round((data[2,k] + data[3,k])/3.0);
   END;
 END;
 draw_background;
 selectworld(6);
 setheaderoff;
 FOR i: = 0 TO 3 DO
  BEGIN
   offset1: = i*625;
   FOR j: = 1 TO 311 DO
    BEGIN
     selectwindow(7);
     draw_wave(i*3 + 1,j,offset1);
     selectwindow(5);
     draw_wave(i*3 + 2,j,offset1);
     selectwindow(6);
     draw_wave(i*3 + 3,j,offset1);
    END;
  END;
 selectwindow(8);
 storewindow(8);
 current_screen: = 8;
END;

{————————————————————————}

OVERLAY PROCEDURE get_rhythm_data;

BEGIN
END;
{————————————————————————}
OVERLAY PROCEDURE display_12_lead;

BEGIN
 restorewindow(8,0,0);
END;

{————————————————————————}
OVERLAY PROCEDURE display_rhythm_strip;

BEGIN
END;

{————————————————————————}
OVERLAY PROCEDURE display_measurements;

BEGIN
END;


{————————————————————————}
{ These procedures control the zoom mode functions          }
{————————————————————————}

OVERLAY PROCEDURE
 zoom2(VAR
z1,z2,z3,z4:INTEGER;lead_number:INTEGER;title string20);

 VAR i,offset,zchoice:INTEGER;

BEGIN
 setforegroundcolor(yellow);
 selectwindow(10);
 current_screen: = 10;
 selectworld(10);
 setbackground(0);
 setheaderOn;
 defineHeader(10,title);
 drawborder;
 FOR i: = 1 TO 311 DO draw_wave(lead_number,i,0);
 storewindow(10);
 CASE lead_number OF
 1,2,3:zoom_offset: = 0;
 4,5,6:zoom_offset: = 625;
 7,8,9:zoom_offset: = 1250;
 10,11,12:zoom_offset: = 1775;
 END;{case}
 setlinestyle(85);
 zchoice: = old_choice MOD 10;
 pull_choice: = old_choice;
 c1: = c1-zoom_offset;
 c2: = c2-zoom_offset;
 c3: = c3-zoom_offset;
 c4: = c4-zoom_offset;  {the origin moved to the beginning of
the lead enlarged}
 CASE zchoice OF
 0:BEGIN
   END;
 1:caliper1(c1,key);
 2:caliper2(c1,c2,key);
 3:caliper3(c1,c2,c3,key);
 4:caliper4(c1,c2,c3,c4,key);
 5:cal_1_2(c1,c2,key);
 6:cal_3_4(c1,c2,c3,c4,key);
 7:cal_all(c1,c2,c3,c4,key);
 END {case};
END;
```

```
OVERLAY PROCEDURE unzoom;

BEGIN
  c1: = c1 + zoom_offset;
  c2: = c2 + zoom_offset;
  c3: = c3 + zoom_offset;
  c4: = c4 + zoom_offset;   {the origin moved back}
  setlinestyle(0);
  setforegroundcolor(white);
  current_screen: = 8;
  clearscreen;
  restorewindow(8,0,0);
END;

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = =
=                                              =
=                MAIN PROGRAM                          =
=                                              =
= = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = =}
{$R+}


 BEGIN
 {if first_time then} initgraphic
        { else entergraphic};
 set_defaults;
 get_12lead_data;
 REPEAT
  Menu(old_choice,pull_choice);
  key: = 'a';
  CASE pull_choice OF
  11:caliper1(c1,key);
  12:caliper2(c1,c2,key);
  13:caliper3(c1,c2,c3,key);
  14:caliper4(c1,c2,c3,c4,key);
  15:cal_1_2(c1,c2,key);
  16:cal_3_4(c1,c2,c3,c4,key);
  17:cal_all(c1,c2,c3,c4,key);

  21:unzoom;

  31:zoom2(c1,c2,c3,c4,1,'Std lead I');
  32:zoom2(c1,c2,c3,c4,2,'Std lead II');
  33:zoom2(c1,c2,c3,c4,3,'Std lead III');
  34:zoom2(c1,c2,c3,c4,4,'avl');
  35:zoom2(c1,c2,c3,c4,5,'avr');
  36:zoom2(c1,c2,c3,c4,6,'avf');
  41:zoom2(c1,c2,c3,c4,7,'V 1');
  42:zoom2(c1,c2,c3,c4,8,'V 2');
  43:zoom2(c1,c2,c3,c4,9,'V 3');
  44:zoom2(c1,c2,c3,c4,10,'V 4');
  45:zoom2(c1,c2,c3,c4,11,'V 5');
  46:zoom2(c1,c2,c3,c4,12,'V 6');
  51:get_12lead_data;
  52:get_rhythm_data;
  55:display_12_lead;
  54:display_rhythm_strip;
  53:;
  61:exit;
  71:print_screen;
  4:display_measurements;
  END; {case}
 UNTIL (pull_choice = 61);
 leavegraphic;
 END.
```

```
{————————————————————————————————}
{   This program is based on an expert system shell written in
}
{   UCSD pascal by Stephen Harris. It has been modified into   .
}
{   Turbo Pascal, and further modifications and additions have
been    }
{   made in customising the program for the present purpose.
}
{                                                   }
{————————————————————————————————}

PROGRAM  Expert_System;

{————————————————————————————————}
{              DATA STRUCTURES.                  }
{                                               }
{    note: stacks are arrays of pointers.            }
{       answers to "questions" stored in a linked list.      }
{————————————————————————————————}

CONST
 MaxNoOfQ = 200;
 MaxNoOfD = 200;
 MaxNoOfR = 300;
 MaxReasons = 100;

 ScreenLength = 14;

TYPE
 twine     =STRING[80];
 whole      =0..maxint;
 textual    = ^packstring;
 packstring = RECORD
  s:STRING[80];
 END;

 dstatus    = (undecided, pending, dfor, dagainst);
 dkinds     = (intermediate,goal);
 dentry     = 1..MaxNoOfd;
 drecord    = RECORD
  TEXT  : textual;
  status : dstatus;
  kind  : dkinds;
 END;

 aentry     = ^arecord;
 arecord    = RECORD
  TEXT : textual;
  next : aentry;
 END;
qentry      = 1..MaxNoOfq;
qrecord     = RECORD
 TEXT  : textual;
 reason : textual;
 first  : aentry;
 answer : whole;
END;

ctypes      = (and_op,or_op,not_op,ques_op,decn_op);
centry      = ^crecord;
crecord     = RECORD
 CASE ctype:ctypes OF
 and_op,or_op:(lhs,rhs:centry);
 not_op     :(operand:centry);
 ques_op    :(ques  :qentry;answ  :whole );
 decn_op    :(decn  :dentry);
END;

rentry      = 1..MaxNoOfr;
rrecord     = RECORD
 condition:centry;
 reason   :textual;
 decision : dentry;
END;

stackrange  = 0..MaxReasons;

charset     = SET OF CHAR;

namestring  = STRING[20];

{= = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= =}
VAR

 Expert:TEXT;

 datastructures: ^CHAR;
 question    : ARRAY[qentry] OF ^qrecord;
 decision    : ARRAY[dentry] OF ^drecord;
 rule      : ARRAY[rentry] OF ^rrecord;
 memory        : whole;  { memory in bytes taken up by the
data }

 aborted,
 badinput,not_opFlag,
 evalactive,undo : BOOLEAN;

 whystack   : ARRAY[stackrange] OF textual;
 stackp     : stackrange;
 ques_stack : ARRAY[stackrange] OF qentry;
 stackp2    : stackrange;
 decn_stack : ARRAY[stackrange] OF dentry;
 stackp3    : stackrange;
```

```
digits,alphabet : charset;

bell:CHAR;

{-----------------------------------}
{     I/O error checking routine and error message routine
}
{                                        }
{-----------------------------------}

PROCEDURE abort(s:twine; i:INTEGER);   FORWARD;

{-----------------------------------}
PROCEDURE clearscreen;

BEGIN
{$I-}
 window(1,1,80,24);
 clrscr;
 window(10,1,80,24);
 writeln;
 IF IOresult0 THEN BEGIN
   textcolor(lightblue);
   writeln('   Sorry, I wasn''t paying attention');
   writeln;
   textcolor(white);
   writeln('Please run me again');
   halt;
  END;
{$I+}
END;

{==================================================================
===============}
{                                        }
{  Text storage & retrieval routines.  If memory availibility
problems }
{  occur they can be alleviated by modifying these routines to
store }
{  the text in a file.                   }
{                                        }
{==================================================================
===============}

PROCEDURE storetext(str:twine ;VAR TEXT:textual );

BEGIN
 new(TEXT);
 TEXT^.s: = str;
 memory: = memory + sizeof(str);
END;

{-----------------------------------}

PROCEDURE writetext(t:textual);

BEGIN
 IF tNIL THEN write(t^.s);
END;

{-----------------------------------}

PROCEDURE writelntext(t:textual);

BEGIN
 IF t = NIL THEN writeln
 ELSE writeln(t^.s);
END;

{-----------------------------------}
PROCEDURE press_any_key;

 VAR bucket:CHAR;

BEGIN
 writeln;
 writeln('Press any key to continue.');
 read(kbd,bucket);
END;

{-----------------------------------}
FUNCTION press_one_of( s:charset ):CHAR;

 VAR ch:CHAR;

BEGIN
 REPEAT
  read(kbd,ch);
  IF (ch = 'a') AND (ch'z') THEN
ch: = chr(ord(ch)-ord('a')+ord('A'));
 UNTIL ch IN s;
 write(ch);
 press_one_of: = ch;
END;

{-----------------------------------}
{                                        }
{     Procedures to handle the reason-stack.          }
{                                        }
{-----------------------------------}

PROCEDURE push(reason:textual);

BEGIN
 IF stackp = MaxReasons THEN
  abort(
  'REASON STACK OVERFLOW. Probably bug in shell.
MaxReasons = ',MaxReasons);
 whystack[stackp]: = reason;
 stackp: = stackp + 1;
```

```
END; { push }

{-------------------------------}

PROCEDURE pop;

BEGIN
 IF stackp0 THEN stackp: = stackp-1
 ELSE abort('Bug in shell: why-stack popped too often.',0);
END;


{-------------------------------}
PROCEDURE push_question(q:whole);

BEGIN
 ques_stack[stackp2]: = q;
 stackp2: = stackp2 + 1;
END; { push }

{-------------------------------}

FUNCTION pop_question(q:whole) qentry;

BEGIN
 IF stackp20 THEN BEGIN
    stackp2: = stackp2-1;
    pop_question: = ques_stack[stackp2];
  END
 ELSE pop_question: = q;
END;

{-------------------------------}

PROCEDURE push_decision(d:whole);

BEGIN
 decn_stack[stackp3]: = d;
 stackp3: = stackp3 + 1;
END; { push }

{-------------------------------}

FUNCTION pop_decision:dentry;

 VAR d:dentry;

BEGIN
 IF stackp30 THEN BEGIN
    stackp3: = stackp3-1;
    pop_decision: = decn_stack[stackp3];
  END;
END;

{-------------------------------}
{                               }
{    Procedures to print out the stored data structures.
}
{                               }
{-------------------------------}

PROCEDURE writeindecn(d dentry);

BEGIN
 WITH decision[d]^ DO BEGIN
   IF kind = intermediate THEN write('(');
   writetext(TEXT);
   IF kind = intermediate THEN write(')');
   writeln;
  END;
END;

{-------------------------------}

PROCEDURE printcond(c:centry);

BEGIN
 IF cNIL THEN CASE c^.ctype OF
  and_op : BEGIN
    write('(');
    printcond(c^.lhs);
    write(' & ');
    printcond(c^.rhs);
    write(')');
   END;

  or_op : BEGIN
    write('(');
    printcond(c^.lhs);
    write(' OR ');
    printcond(c^.rhs);
    write(')');
   END;

  not_op : BEGIN
    write(' NOT ');
    write('(');
    printcond(c^.operand);
    write(')');
   END;

  ques_op : write('Q',c^.ques,'A',c^.answ);

  decn_op : write('D',c^.decn);

  END {case};
END;

{-------------------------------}

PROCEDURE printwhy(w:textual);

BEGIN
 IF wNIL THEN BEGIN
   write('  WHY :');
```

```
    writelntext(w);
  END;
END;

{------------------------------------------}

PROCEDURE printrule(r:rentry);

BEGIN
 IF rule[r]NIL THEN BEGIN
   write('R',r,' :');
   printwhy(rule[r]^.reason);
   write('   IF  ');
   printcond(rule[r]^.condition);
   writeln('  THEN DECISION ',rule[r]^.decision,'.');
  END;
 END;


{------------------------------------------}

PROCEDURE printdecision(d:dentry);

BEGIN
 IF decision[d]NIL THEN BEGIN
   write('D',d,' :  ');
   writelndecn(d);
  END;
 END;


{------------------------------------------}

PROCEDURE printquestion(q:qentry);

 VAR a:aentry;

BEGIN
 IF question[q] NIL THEN BEGIN
   IF (screenlength = whereY + 1) THEN press_any_key;
   write('Q',q,' :  ');
   writelntext(question[q]^.text);
   printwhy(question[q]^.reason);
   a: = question[Q]^.first;
   WHILE aNIL DO BEGIN
     write(' ':12);
     writelntext(a^.text);
     a: = a^.next;
    END;
  END;
 END;

{------------------------------------------}

PROCEDURE Dump;

 VAR i      :INTEGER;
```

```
 choice  :CHAR;

BEGIN
 clearscreen;
 writeln(' LISTING OF CURRENT RULE BASE');
writeln('= = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = ');
 writeln;
 FOR i: = 1 TO MaxNoOfd DO BEGIN
   printdecision(i);
  END;
 writeln;
 press_any_key;
 FOR i: = 1 TO MaxNoOfq DO BEGIN
   printquestion(i);
  END;
 writeln;
 press_any_key;
 FOR i: = 1 TO MaxNoOfr DO BEGIN
   printrule(i);
  END;
 writeln;
 press_any_key;
END;

{------------------------------------------}
{   Procedures to read the knowledge base, check for syntax
}
{   errors and store the knowledge base in the data
structures.   }
{------------------------------------------}


PROCEDURE ReadInput;

 TYPE

tokens = (null,s_str,s_num,s_answer,s_and,s_decision,s_end,s_if
,s_not,
  s_ques,s_rule,s_then,s_why,s_or,s_lparen,s_rparen,s_term);
  severities = (nonfatal,fatal);

 VAR token   : tokens;
  p       : whole;
  line    : STRING[80];
  ch      : CHAR;
  endofinfo : BOOLEAN;
  lastnum : whole;
  laststr : STRING[80];

{------------------------------------------}

PROCEDURE ReadAnItem; FORWARD;

{------------------------------------------}
PROCEDURE ReadCh;

{$R-} {Range checking off for speed reasons}
  VAR skip : BOOLEAN;
```

```
BEGIN
 REPEAT
  skip := TRUE;
  IF p<length(line) THEN BEGIN
    ch:=line[p];
    p:=p+1;
    skip := ch IN ['%','!','\'];
   END;
  IF skip THEN IF eof(expert) THEN endofinfo:=TRUE
   ELSE BEGIN
     readln(expert,line);
     write('.');
     p:=1;
    END;
 UNTIL endofinfo OR NOT skip;
 IF ch IN ['a'..'z'] THEN ch:=chr(ord(ch)-ord('a')+ord('A'));
{$R+}
 END; (* ReadCh *)

{-------------------------------------}
 PROCEDURE NextSymbol; FORWARD;

{-------------------------------------}
 PROCEDURE Recover;

 BEGIN
 END;

{-------------------------------------}
 PROCEDURE SayError(msg:line; severity:severities);

  VAR i:whole;
   choice:CHAR;

 BEGIN
  badinput:=TRUE;
  clearscreen;
  writeln(line);
  FOR i:=1 TO p-2 DO write(' ');
  writeln('^');
  writeln(bell,'*** ERROR: ',msg);
  writeln;
  writeln('Press ace to continue reading in, ' any other key to quit');
  read(kbd,choice);
  IF choice<>' ' THEN halt; {program}
  IF severity=fatal THEN recover;
 END;


{-------------------------------------}
 PROCEDURE NextSymbol;

  {----------}
  PROCEDURE ReadNumber;

   VAR toobig:BOOLEAN;

 BEGIN
  lastnum:=0;
  REPEAT
   lastnum:=10*lastnum+ord(ch)-ord('0');
   ReadCh;
   toobig:=lastnum>3000;
  UNTIL toobig OR NOT(ch IN digits);
  IF toobig THEN BEGIN
    SayError('Number too big  Number has been ignored '
    ,nonfatal);
    WHILE ch IN digits DO ReadCh;
   END;
 END; { ReadNumber }


 {----------}
 PROCEDURE ReadStr(term:CHAR);

  VAR oldp:whole;
   short:STRING[1];

 BEGIN
  short:=' ';
  short[1]:=term;
  oldp:=p;
  p:=pos(short,copy(line,p,length(line)-p+1));
  IF p=0 THEN p:=length(line)
  ELSE p:=p+oldp-1;
  IF line[p]=term THEN BEGIN
    laststr:=copy(line,oldp,p-oldp);
    p:=p+1; ReadCh;
   END
  ELSE BEGIN
    ch:=' ';
    SayError(
    'Text string too long. Quote possibly missing.'
    ,fatal);
   END;
 END;


 {----------}
 PROCEDURE ReadAKeyWord;

  VAR full:STRING[8];
   i  :whole;
   error:BOOLEAN;

 BEGIN
  CASE ch OF
  'A':BEGIN token:=s_answer;  full:='ANSWERS';  END;
  'D':BEGIN token:=s_decision;  full:='DECISION';  END;
  'E':BEGIN token:=s_end;     full:='END';     END;
  'I':BEGIN token:=s_if;      full:='IF';      END;
  'N':BEGIN token:=s_not;     full:='NOT';     END;
  'O':BEGIN token:=s_or;      full:='OR';      END;
```

```
    'Q':BEGIN token:=s_ques;    full:='QUESTION';  END;
    'R':BEGIN token:=s_rule;    full:='RULE';      END;
    'T':BEGIN token:=s_then;    full:='THEN';      END;
    'W':BEGIN token:=s_why;     full:='WHY';       END;
     'B','C','F','G','H'
     ,'J','K','L','M'
     ,'P','S','U','V'
    ,'X','Y','Z'     :     full:='';
    END; { case }
    i:=1; error:=FALSE;
    WHILE (ch IN alphabet) AND NOT error
     DO BEGIN
      IF ilength(full) THEN error:=TRUE
      ELSE IF full[i]<>ch THEN error:=TRUE
      ELSE BEGIN ReadCh; i:=i+1; END;
     END;
    IF error THEN
     BEGIN
      IF full<>'' THEN
       SayError('Illegal keyword. Has been ignored',nonfatal)
      ELSE SayError(concat
       ('Unrecognised keyword. "',full,'" assumed.'),
       nonfatal);
      WHILE ch IN alphabet DO ReadCh;
     END;
   END; { ReadAKeyWord }


BEGIN { nextsymbol }
 token:=null;
 REPEAT
  IF endofinfo THEN token:=s_end
  ELSE
   IF ch IN ['0'..'9', '"', '''', ' ', '+', '-',
    '=', '*', '(', ')', '&', 'A'..'Z']
    THEN BEGIN
     CASE ch OF
      ' ','+','-','=','*' : ReadCh;
      '"','''      : BEGIN
        ReadStr(ch);
        token:=s_str;
       END;
      '&'          : token:=s_and;
      '('          : token:=s_lparen;
      ')'          : token:=s_rparen;
      '.'          : token:=s_term;
       '0','1','2','3','4','5',
      '6','7','8','9' : BEGIN
        ReadNumber;
        token:=s_num;
       END;
       'A','B','C','D','E','F','G','H'
      ,'I','J','K','L','M','N','O','P'
      ,'Q','R','S','T','U','V','W','X'
      ,'Y','Z'                  : ReadAKeyWord;
     END; { case }
      IF token IN [s_and,s_lparen,s_rparen,s_term]
       THEN ReadCh;
     END
    ELSE BEGIN
      SayError('Illegal character ignored',nonfatal);
      ReadCh;
     END;
  UNTIL tokennull;

 END; { NextSymbol }


{-------------------------------------}

 PROCEDURE Expect(atoken:tokens; msg:twine);

 BEGIN
  IF token=atoken THEN NextSymbol
  ELSE SayError(msg,fatal);
 END;

{-------------------------------------}
 PROCEDURE Ignore(it:tokens);

 BEGIN
  IF it=token THEN NextSymbol;
 END;

{-------------------------------------}
 FUNCTION Expectnum(max:whole):whole;

  VAR number:whole;

 BEGIN
  number:=lastnum;
  expect(s_num,'Number expected');
  IF (number=1) AND (number<=max)
   THEN expectnum:=number
  ELSE Sayerror('Number out of range.',fatal);
 END;

{-------------------------------------}
 FUNCTION decnnumber:dentry;

 BEGIN
  decnnumber:=expectnum(MaxNoOfd);
 END;

{-------------------------------------}
 FUNCTION quesnumber:qentry;

 BEGIN
  quesnumber:=expectnum(MaxNoOfq);
 END;

{-------------------------------------}
 FUNCTION rulenumber:rentry;
```

```
BEGIN
 rulenumber:=expectnum(MaxNoOfr);
END;

{-------------------------------}

PROCEDURE ReadStoStr(VAR TEXT:textual;errormsg:twine
;severity:seventies);

BEGIN
 IF token=s_str THEN BEGIN
   storetext(laststr,TEXT);
   NextSymbol;
  END
 ELSE
  IF severity=fatal
   THEN SayError(errormsg,fatal)
  ELSE TEXT:=NIL;
END;

{-------------------------------}
PROCEDURE ReadWhy(VAR TEXT:textual);

BEGIN
 TEXT:=NIL;
 IF token=s_why THEN BEGIN
   NextSymbol;
   ReadStoStr(TEXT,'String expected',fatal);
  END;
END;

{-------------------------------}
PROCEDURE ReadDecision;

 VAR number:whole;

BEGIN
 NextSymbol;
 number:=decnnumber;
 IF decision[number]=NIL
  THEN BEGIN
   new(decision[number]);
   WITH decision[number]^
    DO BEGIN
     status:=undecided;
     IF token = s_lparen
      THEN BEGIN
       NextSymbol;
       ReadStoStr(TEXT,'',nonfatal);
       expect(s_rparen,
       ' ")" expected after "(
s-"');
       kind:=intermediate;
      END
     ELSE BEGIN
       ReadStoStr(TEXT,'',nonfatal);
       IF TEXT=NIL
        THEN kind:=intermediate ELSE kind:=goal;
      END;
    END
  END
 ELSE Sayerror('Multiple decision definition.',fatal);
END;

{-------------------------------}
PROCEDURE ReadQuestion;

 VAR number:qentry;

  last :aentry;

 {-------------}
 FUNCTION answer_rec:aentry;

  VAR ans:aentry;

 BEGIN
  new(ans);
  ans^.next:=NIL;
  ignore(s_num);
  ReadStoStr(ans^.text,'Answer option text expected',fatal);
  answer_rec:=ans;
 END;


BEGIN
 NextSymbol;
 number:=quesnumber;
 IF question[number]=NIL
  THEN BEGIN
   new(question[number]);
   WITH question[number]^
    DO BEGIN
     first:=NIL;
     answer:=0;
     ReadStoStr(TEXT,
     'A question needs text to ask.',fatal);
     readwhy(reason);
     expect(s_answer,'"answer" expected');
     first:=answer_rec;
     last:=first;
     WHILE token IN [s_num,s_str]
      DO BEGIN
       last^.next:=answer_rec;
       last:=last^.next;
      END;
    END
  END
 ELSE BEGIN
   Sayerror('Multiple question definition.',fatal);
  END;
END;

{-------------------------------}
```

```
PROCEDURE ReadRule;

 VAR number:INTEGER;
  rulerec: ^rrecord;

 {————————}
 FUNCTION or_tree:centry;

 FORWARD;

 {————————}
 FUNCTION factor:centry;

  VAR c:centry;

 BEGIN
  IF token IN [s_lparen,s_ques,s_decision]
   THEN CASE token OF
   s_lparen  :BEGIN NextSymbol;
     c: = or_tree;
     expect(s_rparen,
     ' ")", "&" or "OR"  expected.');
    END;
   s_ques   :BEGIN NextSymbol;
     new(c);
     c ^ .ctype: = ques_op;
     c ^ .ques: = quesnumber;
     ignore(s_answer);
     c ^ .answ: = expectnum(9);
    END;
   s_decision :BEGIN NextSymbol;
     new(c);
     c ^ .ctype: = decn_op;
     c ^ .decn: = decnnumber;
    END;
   END  { case }

   ELSE SayError(' "(", "QUESTION" or "DECISION"
expected',
    fatal);
  factor: = c;
 END; {  func factor }

 {————————}
 FUNCTION not_tree:centry;

  VAR c:centry;

 BEGIN
  IF token = s_not THEN BEGIN
    NextSymbol;
    new(c);
    c ^ .ctype: = not_op;
    c ^ .operand: = not_tree;
   END
  ELSE c: = factor;
  not_tree: = c;
 END; { not_tree}

 {————————}
 FUNCTION and_tree:centry;

  VAR c,c1:centry;

 BEGIN
  c: = not_tree;
  IF token = s_and THEN BEGIN
    NextSymbol;
    c1: = c;
    new(c);
    c ^ .ctype: = and_op;
    c ^ .lhs: = c1;
    c ^ .rhs: = NIL; {secure bomb}
    c ^ .rhs: = and_tree;
   END;
  and_tree: = c;
 END;  { and_tree }

 {————————}
 FUNCTION or_tree{:centry};

  VAR c,c1:centry;

 BEGIN
  c: = and_tree;
  IF token = s_or  THEN BEGIN
    NextSymbol;
    c1: = c;
    new(c);
    c ^ .ctype: = or_op;
    c ^ .lhs: = c1;
    c ^ .rhs: = NIL; {secure bomb}
    c ^ .rhs: = or_tree;
   END;
  or_tree: = c;
 END;  { or_tree }

BEGIN  { read rule }
 NextSymbol;
 number: = rulenumber;
 IF rule[number]NIL THEN
  Sayerror('Multiple rule definition.',fatal);
 new(rulerec);
 WITH rulerec ^  DO BEGIN
   condition: = NIL; {secure if bombs}
   ReadWhy(reason);
   expect(s_if,'"IF"  (or "WHY")  expected.');
   condition: = or_tree;
   expect(s_then,
   '"THEN"  (or "&" or "OR")    expected.');
   expect(s_decision,'"DECISION" expected.');
   decision: = decnnumber;
  END;
 rule[number]: = rulerec;
```

```
END; { read rule }


{-------------------------------------}
PROCEDURE ReadAnItem;

BEGIN
 CASE token OF
  s_str,s_num,s_answer,s_and,s_if,
  s_not,s_then,s_why,s_lparen,
 s_rparen,s_or              :BEGIN
    SayError(
    'Decision,question or rule expected.Ignored',fatal);
    NextSymbol;
   END;

  s_decision                :readdecision;
  s_ques                    :readquestion;
  s_rule                    :readrule;
  s_term                    :NextSymbol;
  s_end                     :;
 END (* case *)
END; (* ReadAnItem *)


{-------------------------------------}
PROCEDURE InitialiseRead;


  TYPE
   string80 = STRING[80];
   TimeRec = RECORD
    TimeComp  : INTEGER;
    TimeString : String80;
    Hours,Minutes,Seconds,Hundredths : INTEGER
   END;
   DateRec = RECORD
    DateComp     : INTEGER;
    DateString   : String80;
    Year,Month,Day : INTEGER;
    DayOfWeek    : INTEGER
   END;
   string15 = STRING[15];
   DIRPtr = ^DIRRec;
   DIRRec = RECORD
    FileName  : String15;
    Attrib    : Byte;
    FileSize  : REAL;
    TimeStamp : TimeRec;
    DateStamp : DateRec;
    Next      : DIRPtr;
   END;

  VAR pointer,baseptr      :dirptr;
   info              :dirrec;
   I,J,lineno,num,result  :INTEGER;
   ch1,ch2           :CHAR;
   botch,drivechanged    :BOOLEAN;
   knowl_base,oldfilename :STRING[15];
   drive,newdrive        :STRING[2];
   heaptop           :^INTEGER;

                                          .
{                                         }
{   The following function accesses the disk directory and
}{
{   returns the extended directory listing as a linked list     }
{                                         }



FUNCTION DynaDIR(Filespec : String80) : DIRPtr;

 TYPE
  String9 = STRING[9];
  Reg    = RECORD
   CASE BOOLEAN OF
   FALSE : (Word : INTEGER);
   TRUE  : (LoByte,HiByte : Byte)
  END;

  Regpack = RECORD
   AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Reg
  END;

  DWord   = RECORD
   LoInteger,HiInteger : INTEGER
  END;


 VAR
  I      : INTEGER;
  Registers : RegPack;
  Root     : DIRPtr;
  Current  : DIRPtr;
  Prior    : DIRPtr;
  ASCIIZ   : ARRAY[1..81] OF CHAR;

  {-----------------------}
  PROCEDURE DTAtoDIR(VAR OutRec : DIRRec);

  CONST
   MonthTags : ARRAY [1..12] OF String9 =
   ('January','February','March','April','May','June','July',
   'August','September','October','November','December');
   DayTags  : ARRAY [1..7] OF String9 =
   ('Sunday','Monday','Tuesday','Wednesday',
   'Thursday','Friday','Saturday');

  TYPE
   String5 = STRING[5];
   Dword   = RECORD
    LoInteger,HiInteger : INTEGER
   END;
```

```
    DTAPtr = ^DTARec;
    DTARec = RECORD
     Reserved : ARRAY[0..20] OF Byte;
     Attrib  : Byte;
     TimeComp : INTEGER;
     DateComp : INTEGER;
     FileSize : DWord;
     FileName : ARRAY[1..13] OF CHAR
    END;

   VAR
    I       : INTEGER;
    Temp1,Temp2 : String5;
    AMPM     : CHAR;
    InRec    : DTARec;
    Registers  : Regpack;
    CurrentDTA  : DTAPtr;

   BEGIN
    Registers.AX.Word := $2F00; { Find current location of
DTA }
    MSDOS(Registers);
    WITH Registers DO CurrentDTA := Ptr(ES.Word,BX.Word);
    InRec := CurrentDTA^;
    WITH OutRec DO          { Now extract and reformat data
}
     BEGIN
      I := 1;            { Extract the file name field }
      WHILE InRec.FileName[I] Chr(0) DO
       BEGIN
        FileName[I] := InRec.FileName[I];
        I := Succ(I)
       END;
      FileName[0] := CHR(I-1);
      Attrib := InRec.Attrib; { Extract the attribute field }
      Next := NIL;          { Initialize the "next" pointer }
     END
   END;


   BEGIN
    dynadir:=NIL;
  {First step is to convert Filespec string to ASCIIZ:}
    Filespec := Filespec + CHR(0);  { Append binary zero to
Filespec }
    Move(Filespec[1],ASCIIZ,Sizeof(Filespec));
    WITH Registers DO
     BEGIN
      AX.Word := $4E00;     { $4E = Find First }
      DS.Word := Seg(ASCIIZ); { Put address of ASCIIZ }
      DX.Word := Ofs(ASCIIZ); { in DS : DX }
     END;
    MSDOS(Registers);          { Make FIND FIRST DOS call... }
    IF Registers.AX.Word = 2 THEN DynaDIR := NIL
    ELSE
     BEGIN
      New(Root);          { Convert first find to DIR format }
      DTAtoDIR(Root^);
      Prior := Root;
    IF Registers.AX.Word  18 THEN
     REPEAT
      Registers.AX.Word := $4F00;
      MSDOS(Registers);  { Make FIND NEXT DOS call... }
      IF Registers.AX.Word  18 THEN
       BEGIN
        New(Current);
        DTAtoDIR(Current^); { Convert additional finds }
        Prior^.Next := Current; { to DIRRec format }
        Prior := Prior^.Next
       END
     UNTIL Registers.AX.Word = 18;
    DynaDIR := Root;
   END
  END; {DynaDIR}


{-------------------------------------}

 BEGIN
{$i-}
  memory := sizeof(question) + sizeof(decision) + sizeof(rule);
  botch. = FALSE;
  close(Expert);
  getdir(0,drive);
  drivechanged. = FALSE;
  REPEAT
   clearscreen;
   writeln;
   getdir(0,newdrive);
   writeln('These are the question sets on the ',newdrive,'
drive');
   writeln;
   writeln('Please choose the question set you are interested
in:- ');
   writeln;
   textcolor(yellow);
   write('Q':10);
   textcolor(white);
   write(' - ');
   writeln('to Quit');
   writeln;
   pointer:=baseptr;
   baseptr:=NIL;
   mark(heaptop);
   baseptr:=dynaDIR('*.txt');
   IF (drivechanged) AND (pointer=baseptr) THEN
baseptr:=NIL;
   pointer:=baseptr;
   j:=0;
   WHILE pointerNIL DO
    BEGIN
     WITH pointer^ DO
      BEGIN
       i:=pos('.',filename);
       delete(filename,i,4);
       j:=j+1;
       textcolor(yellow);
       write(j:10);
```

```
        textcolor(white);
        write(' - ');
        writeln(filename);
      END;
    pointer: = pointer ^ .next;
  END;
 IF j = 0 THEN BEGIN
   textcolor(yellow);
   writeln(' There are no question sets on this drive ');
   textcolor(white);
  END;
 writeln;
 lineno: = wherey;
 REPEAT
  gotoxy(0,lineno);write(' ':80);
  gotoxy(0,lineno);
  write('Please select your choice ( 1 to 'j,' ) or (A,B,C) to
change drive ');
  ch1: = '0';
  ch2: = '0';
  read(kbd,ch1);write(ch1);
  delay(800);
  IF keypressed THEN BEGIN
    read(kbd,ch2);
    write(ch2);
   END
  ELSE BEGIN
    ch2: = ch1;
    ch1: = '0';
   END;
  val(ch1 + ch2,num,result);
  writeln;
  drivechanged: = FALSE;
  IF ch2 IN ['q','Q'] THEN BEGIN
    chdir(drive);
    halt;
   END;
  IF ch2 IN ['a','b','c','C','B','A']
   THEN BEGIN
    ch2: = upcase(ch2);
    IF (ch2) newdrive
     THEN BEGIN
      release(heaptop);
      mark(heaptop);
      drivechanged: = TRUE;
      newdrive: = ch2 + ':';
      ChDir(newdrive);
      getdir(0,newdrive);
      IF newdrivech2 + ':' THEN
       drivechanged: = FALSE;
     END;
   END;
 UNTIL ((num  1) AND (num0) AND (result = 0)) OR
drivechanged;
 IF NOT drivechanged
  THEN BEGIN
   pointer: = baseptr;
   i: = 1;
   WHILE I_DO BEGIN
     pointer: = pointer ^ .next;
     i: = i + 1;
    END;
   knowl_base: = pointer ^ .filename;
   assign(expert,(newdrive + knowl_base + '.txt'));
   reset(Expert);
   botch: = ioresult0;
   IF botch THEN BEGIN
     writeln;textcolor(lightred);
     writeln('I cannot find "',knowl_base,'".');
     delay(2000); textcolor(white);
    END;
  END;
 UNTIL (NOT botch) AND (NOT drivechanged);
 release(heaptop);
{$i + }
 chdir(drive);        {go back to the old drive}
 line: = '';
 p: = 2;
 endofinfo: = FALSE;
 ReadCh;
 NextSymbol;
END;

{————————————————————————————}

BEGIN  (* Read Input main procedure *)
 InitialiseRead;
 REPEAT
  ReadAnItem;
 UNTIL token = s_end;
END;

{————————————————————————————————————————}
{     The following procedures check (audit) the stored
knowledge     }
{     for logical and declarative errors.                    }
{                                                            }
{————————————————————————————————————————}

PROCEDURE  Audit;


 TYPE bloggs = SET OF 1..200;

 VAR  d_ref,d_dcl:bloggs;  { decisions referenced and
declared }
  q_ref,q_dcl:SET OF qentry;
  d:dentry;
  q:qentry;
  r:rentry;


 {————————————————————}
 PROCEDURE  Audit_condition( cond:centry );

 BEGIN
```

```
 IF condNIL THEN WITH cond^ DO
   CASE otype OF
   and_op,or_op: BEGIN
     Audit_condition(lhs);
     Audit_condition(rhs);
    END;
   not_op    : Audit_condition(operand);
   ques_op   : q_ref:=q_ref+[ques];
   decn_op   : d_ref:=d_ref+[decn];
   END; { case }
END; { audit condition }

{-------------------------}
PROCEDURE Audit_error;

 VAR  d:dentry;      q:qentry;

 BEGIN
  badinput:=TRUE;
  clearscreen;
  writeln(
  'AUDIT *** The following anomalies have been found in
the question set');
  writeln(
  '-----------------------------------');
  writeln;
  IF (d_dcl-d_ref)[]
   THEN BEGIN
    writeln('The following decisions are declared but not
used:');
    FOR d:=1 TO MaxNoOfd DO
     IF d IN (d_dcl-d_ref) THEN write(d,' ');
    writeln;
   END;
  IF (q_dcl-q_ref)[]
   THEN BEGIN
    writeln('The following questions are declared but not
used:');
    FOR q:=1 TO MaxNoOfq DO
     IF q IN (q_dcl-q_ref) THEN write(q,' ');
    writeln;
   END;
  IF (d_ref-d_dcl)[]
   THEN BEGIN
    writeln('The following decisions are not declared !');
    FOR d:=1 TO MaxNoOfd DO
     IF d IN (d_ref-d_dcl) THEN write(d,' ');
    writeln;
   END;
  IF (q_ref-q_dcl)[]
   THEN BEGIN
    writeln('The following questions are not declared !');
    FOR q:=1 TO MaxNoOfq DO
     IF q IN (q_ref-q_dcl) THEN write(q,' ');
    writeln;
   END;
  press_any_key;
 END;

BEGIN
 d_dcl:=[]; d_ref:=[]; q_dcl:=[]; q_ref:=[];
 FOR d:=1 TO MaxNoOfd DO
  IF decision[d]NIL THEN d_dcl:=d_dcl+[d];
 FOR q:=1 TO MaxNoOfq DO
  IF question[q]NIL THEN q_dcl:=q_dcl+[q];
 FOR r:=1 TO MaxNoOfr DO
  IF rule[r]NIL
   THEN BEGIN
    d_ref:=d_ref+[rule[r]^.decision];
    Audit_condition(rule[r]^.condition);
   END;
 IF (d_dcld_ref) OR (q_dclq_ref) THEN Audit_error;
END; { audit }

{-----------------------------------}

PROCEDURE ReadInteger( line:whole; prompt:twine ;VAR
n:whole; max:whole );

 VAR ch    : CHAR;
  valid : BOOLEAN;
  reply : STRING[80];
  i     : INTEGER;

BEGIN
{$I-}
 REPEAT
  gotoxy(0,line);write(' ':80);
  gotoxy(0,line+1);write(' ':80);
  gotoxy(0,line);write(prompt);
  readln(n);
  valid:=(ioresult=0) AND (1<=n) AND (n<=max);
  IF NOT valid THEN
   BEGIN
    writeln;
    writeln(bell,'I am expecting a number between 1 and
',max,'.');
   END;
 UNTIL valid;
{$I+}
END; { ReadInteger }


{-----------------------------------}
FUNCTION answerto(q:qentry;expected:whole):whole;

 VAR a        :aentry;
  i,why_line,step  :whole;
  whyp       :stackrange;
  ch         :CHAR;
  answered   :BOOLEAN;

BEGIN
 IF question[q]=NIL THEN abort('There is no question ',q);
 WITH question[q]^ DO BEGIN
   IF (answer=0) AND evalactive
    THEN BEGIN
```

```
clearscreen;
writeln('Question:');
writelntext(TEXT);
a: = first;
i: = 0;
WHILE aNIL DO BEGIN
  i: = i + 1;
  write(i:5,') ');
  writelntext(a^.text);
  a: = a^.next;
 END;
push(reason);
answered: = FALSE;
whyp: = stackp;
why_line: = i + 8;
step: = 2;
writeln;
write(' Press');
textcolor(yellow);write(' W');
textcolor(white);write(' or ');
textcolor(yellow);write(' ?');
textcolor(white);write(' - for explanation. ');
textcolor(yellow);write(' H ');
textcolor(white);writeln(' - for a hint. ');
write(' Press ');
textcolor(yellow);write('0    ');
textcolor(white);
writeln(' - to go back to previous question ');
writeln;
REPEAT
 gotoxy(0,why_line + 2);write(' ':80);
 gotoxy(0,why_line + 2);
 write(' Press your selection ');
 ch: = press_one_of
 (['1'..chr(ord('0') + i),'W','?','0','H']);
 IF ch IN ['W','?']
  THEN BEGIN
   REPEAT
    IF whyp0 THEN whyp: = whyp-1
   UNTIL whystack[whyp]NIL;
   IF why_line
gth-5
    THEN why_line: = why_line + step;
   step: = 1;
   gotoxy(0,why_line + 1);write(' ':80);
   gotoxy(0,why_line);write(' ':80);
   gotoxy(0,why_line);
   writelntext(whystack[whyp]);
  END
 ELSE IF ch = 'H'
  THEN BEGIN
   write('Int - ');
   IF not_opFlag THEN write(' avoid ')
   ELSE write(' try ');
   textcolor(yellow);
   write(expected);
   delay(1000);
   textcolor(white);
   END
  ELSE answered: = TRUE;
  IF ch = '0' THEN undo: = TRUE
  ELSE BEGIN
    undo: = FALSE;
    push_question(q);
   END;
 UNTIL answered;
 answer: = ord(ch)-ord('0');
 pop;
END;
answerto: = answer;
END; { with }
END; { Answer_to }


{————————————————}
FUNCTION decide(decn:dentry):BOOLEAN;

FORWARD;

{————————————————————————————————}
{   Returns the value of the supplied expression. Recursion
is  }
{   used to evaluate the more complex expression trees.
}{                                                  }
{————————————————————————————————}

FUNCTION Satisfied(cond:centry):BOOLEAN;

BEGIN
 IF NOT undo THEN
  WITH cond^ DO
   BEGIN
    CASE cond^.ctype OF
    and_op :IF satisfied(lhs) THEN satisfied: = satisfied(rhs)
     ELSE satisfied: = FALSE;
    or_op  :IF satisfied(lhs) THEN satisfied: = TRUE
     ELSE satisfied: = satisfied(rhs);
    not_op :BEGIN
      not_opFlag: = TRUE;
      satisfied: = NOT satisfied(operand);
      not_opFlag: = FALSE;
     END;
    ques_op:satisfied: = (answerto(ques,answ) = answ);
    decn_op:satisfied: = (decide(decn));
    END;
   END
 ELSE satisfied: = FALSE; {default}
END; { satisfied }

{————————————————————————————————}
{   The DECIDE function returns TRUE if the supplied
decision is  }
{   valid, FALSE if not valid                       }
{   and aborts if there are no rules to determine the decision
}{
{   (This should never happen: proc Audit should detect the
error) }
```

```
{                                                    }
{------------------------------------------------------}

FUNCTION {decide(decn:dentry):boolean; Declared forward.}

 VAR ruleno:0..MaxNoOfr;
  satisfaction,found:BOOLEAN;

BEGIN
 push_decision(decn);
 WITH decision[decn]^ DO
  CASE status OF
  pending  : abort('Cyclic rules for decision ',decn);
  dfor     : decide:= TRUE;
  dagainst : decide:= FALSE;
  undecided : BEGIN
     status:= pending;
     satisfaction:= FALSE;
     found:= FALSE;
     ruleno:= 0;
     WHILE (ruleno<MaxNoOfr) AND NOT satisfaction DO
      BEGIN
       ruleno:= ruleno + 1;
       IF rule[ruleno]<>NIL
        THEN WITH rule[ruleno]^ DO
         IF decision = decn
          THEN BEGIN
           push(reason);
           satisfaction:= satisfied(condition);
           pop;
           found:= TRUE;
          END;
      END;
     IF NOT found
      THEN abort(
      'Insufficient information to validate decision ',decn);
     decide:= satisfaction;
     IF satisfaction THEN status:= dfor
     ELSE status:= dagainst;
     IF undo THEN status:= undecided;
    END;
  END; { case }
END; { decide }


{------------------------------------------------------}
{    The procedures reset_memory and reset_last 'forget' the answers }
{    already, to all questions or the most recently asked question }
{    respectively.                                     }
{------------------------------------------------------}

PROCEDURE Reset_memory;

 VAR q:qentry;
  d:dentry;

BEGIN
 FOR q:= 1 TO MaxNoOfq DO
  IF question[q]<>NIL THEN
   question[q]^.answer:= 0;
 FOR d:= 1 TO MaxNoOfd DO
  IF decision[d]<>NIL THEN
   decision[d]^.status:= undecided;
END;


{------------------------------------------------------}
PROCEDURE Reset_last;

 VAR q:whole;
  d:dentry;

BEGIN
 q:= pop_question(q);
 IF question[q]<>NIL THEN
  question[q]^.answer:= 0;
 WHILE stackp3 <> 0 DO
  BEGIN
   d:= pop_decision;
   decision[d]^.status:= undecided;
  END;
END;

{------------------------------------------------------}
{    This procedure lists all the possible decisions that can be reached by the current database.  }
{------------------------------------------------------}

PROCEDURE List_Decisions;

 VAR d      :dentry;
  count     :whole;
  choice    :CHAR;
  again     :BOOLEAN;

BEGIN
 REPEAT
  clearscreen;
  again:= FALSE;
  count:= 0;
  choice:= ' ';
  FOR d:= 1 TO MaxNoOfd DO
   IF decision[d]<>NIL
    THEN BEGIN
     count:= (count + 1) MOD 19;
     IF (count = 0) AND (NOT again) AND (decision[d]^.text<>NIL)
      THEN BEGIN
       writeln;
       write('Press ');
       textcolor(yellow);
       write('<cr>');
       textcolor(white);
```

```
        write(' to continue listing decisions.');
        textcolor(yellow);
        write(' R ');
        textcolor(white);
        writeln(' to repeat the list, ');
        writeln(' or any other key if you are ready to choose.');
        read(kbd,choice);
        IF ((choice = 'R') OR (choice = 'r')) THEN again: = TRUE;
        IF choice = ' ' THEN clearscreen
        ELSE IF NOT again THEN exit;
       END;
      IF (choice = ' ') AND
       (decision[d] ^ .textNIL) THEN BEGIN
        write(d:3,' ');
        writeintext(decision[d] ^ .text);
       END;
     END;
  IF (NOT again)
   THEN BEGIN
    writeln;
    write(' Press ');
    textcolor(yellow);
    write(' R ');
    textcolor(white);
    writeln(' to repeat the list, ');
    writeln(' or any other key if you are ready to choose.');
    read(kbd,choice);
    IF ((choice = 'r') OR (choice = 'R')) THEN again: = TRUE;
   END
  ELSE count: = 1;
 UNTIL (NOT again);
END;



{————————————————————————}
{$I b:graph.p}   {include TURBO extended graphics}


{————————————————————————}
{   This procedure draws a map of the world. It uses two external   }
{   files - line.inv draws very quickly. Worldmap.dat contains the   }
{   map coordinates. The procedure is derived from a public domain   }
{   program written by Jeff Firestone.                               }
{————————————————————————}

PROCEDURE Worldmap;

 CONST
  ArSize = 1280;
 VAR
  n,x,x1,y,y1,i,j : INTEGER;
  WorldArray : ARRAY [0..ArSize] OF INTEGER;
  f : FILE;
  ch:CHAR;

PROCEDURE line(a,b,c,d,e:INTEGER); EXTERNAL 'line.inv';

PROCEDURE ReadInfo;
BEGIN
 FillChar(WorldArray, sizeof(worldarray), 0);
 assign(f, 'worldmap.dat');
{I-}reset(f);{I+}
 IF IOresult0 THEN halt;
 BlockRead(f, WorldArray, round((ArSize/128)*2));
 close(f);
END;


BEGIN
 graphcolormode;
 ReadInfo;
 i: = 0;
 n: = WorldArray[i]; i: = i + 1;
 REPEAT
  x1: = WorldArray[i] DIV 2; i: = i + 1;
  y1: = WorldArray[i]; i: = i + 1;
  x : = WorldArray[i] DIV 2; i: = i + 1;
  y : = WorldArray[i]; i: = i + 1;
  line (x1, y1, x, y, 1);
  x1: = x; y1: = y;
  FOR j: = 3 TO n DO
   BEGIN
    x : = WorldArray[i] DIV 2; i: = i + 1;
    y : = WorldArray[i]; i: = i + 1;
    line (x1, y1, x, y, 1);
    x1: = x; y1: = y;
   END;
  n: = WorldArray[i]; i: = i + 1;
 UNTIL (n  1);
 line(171,72,174,82,3);
 fillshape(171,82,3,1);
 setpencolor(2);
 gotoxy(8,4);write('UNIVERSITY of CAPE TOWN');
 gotoxy(12,6);writeln('Medical School');
 gotoxy(1,24);write('Press any key to start....');
 read(kbd,ch);
 textmode;
END;

{————————————————————————}
{   This procedure controls the animation sequence.   }
{                                                     }
{————————————————————————}

PROCEDURE Animate(number:whole);

 TYPE datum = RECORD
   value:byte;
   times:byte;
  END;
  matrix = ARRAY[0000 .. MAXINT] OF byte;
  bufmat = ARRAY[0..12000] OF datum;
```

```
VAR
 f          :FILE OF bufmat;
 i          :INTEGER;
 j,k,n,m    :INTEGER;
 buffer     :matrix;
 inname     :STRING[9];
 OK         :BOOLEAN;
 index      :CHAR;

{-------------------------------------}

 PROCEDURE expand;

  VAR outbuf      :bufmat;


 BEGIN
  read(f,outbuf);
  close(f);
  k:=outbuf[0].times;
  n:=0000;
  FOR m:= 1 TO k*64 DO
   BEGIN
    fillchar(buffer[n],outbuf[m].times,outbuf[m].value);
    n:=n+outbuf[m].times;
   END;
  putpic(buffer,0,199);
  index:=succ(index);
  assign(f,(inname+'c.'+index));
        {$I-}reset(f);{$I+}
  IF ioresult0 THEN BEGIN
     index:='a';
     assign(f,(inname+'c.'+index));
                    {$I-}reset(f);{$I+}
    END;
 END;

BEGIN
 str(number,inname);
 inname:='d'+inname;
 index:='a';
 assign(f,(inname+'c.'+index));
{$I-}reset(f);{$I+}
 OK:=(IOresult=0);
 graphcolormode;
 graphbackground(blue);
 palette(3);
 IF NOT OK THEN BEGIN
    writeln('ANIMATION FOR THIS DECISION');
    writeln;
    writeln('x   HAS NOT BEEN FOUND');
    delay(2000);
  END
  ELSE REPEAT
    expand
  UNTIL keypressed;
 close(f);
 IF keypressed THEN read(kbd,index);
 textmode;
END;

{-------------------------------------
{  This procedure controls the backward chaining search
strategy  }
{                                     }
{-------------------------------------}


PROCEDURE User_Validate;

 VAR  choice,ch:CHAR;
  decn,i  :whole;
  outcome:STRING[11];
  do_again:BOOLEAN;

BEGIN
 clearscreen;
 BEGIN
  REPEAT
   List_decisions;
   ReadInteger(20,
   ' Enter the decision number (number, plus
),'
   ,decn,10000);
   clearscreen;
  UNTIL ( (decn<=MaxNoOfD) AND (decision[decn]<>NIL) );
  write('Would you like answers to previous questions '
  ,'to be ignored? (Y/N)');
  ch:=press_one_of(['Y','N']);
  IF ch='Y' THEN reset_memory;
  IF decn<=MaxNoOfD
   THEN IF decision[decn]<>NIL
    THEN BEGIN
     REPEAT
      do_again:=FALSE;
      IF decide(decn)
       THEN outcome:='CORRECT'
      ELSE outcome:='NOT CORRECT';
      IF undo THEN BEGIN
        reset_last;
        undo:=FALSE;
        do_again:=TRUE;
       END;
     UNTIL NOT do_again;
     clearscreen;
     writeln('Decision ',decn,':');
     writeln;
     writelntext(decision[decn]^.text);
     write('This interpretation is ',
     outcome,'.');
     writeln;
     press_any_key;
```

```
      IF outcome = 'CORRECT'
       THEN
       BEGIN
        clearscreen;
        writeln;
        write
        ('Would you like tc see this animated? (Y/N)');
        ch: = press_one_of(['Y','N']);
        IF (ch = 'Y')THEN
          BEGIN
            animate(decn);
          END;
       END;
     END;
  END;
END;


{————————————————————————————————————}
{  This procedure controls the pseudo forward chaining
search      }
{  strategy.                                          }
{                                                     }
{————————————————————————————————————}

PROCEDURE Be_an_expert;

 VAR  d:dentry;
  valid,found,extra,do_again:BOOLEAN;
  ch:CHAR;

BEGIN
 reset_memory;
 found: = FALSE;
 extra: = FALSE;
 FOR d: = 1 TO MaxNoOfd DO
  IF decision[d]NIL
   THEN IF decision[d] ^ .textNIL
    THEN BEGIN
     REPEAT
      do_again: = FALSE;
      valid: = decide(d);
      IF undo THEN BEGIN
         reset_last;
         undo: = FALSE;
         do_again: = TRUE;
        END;
     UNTIL NOT do_again;
     IF valid
      THEN BEGIN
        found: = TRUE;
        extra: = FALSE;
        clearscreen;
        writeintext(decision[d] ^ .text);
        writeln;
        write
        ('Would you like to see this animated? (Y/N)');
        ch: = press_one_of(['Y','N']);
        IF (ch = 'Y')THEN
         BEGIN
          animate(d);
         END;
        writeln;
        writeln('Press - ace to search for additional interpretations. ');
        writeln('     - M for main menu.');
        ch. = press_one_of([' ','M']);
        IF ch = 'M' THEN exit;
        extra: = TRUE;
       END;
    END;

 IF  NOT found
  THEN BEGIN
   clearscreen;
   writeln('I was not able to find a valid interpretation.');
  END
 ELSE IF extra THEN BEGIN
   clearscreen;
   writeln('There are no more valid interpretations.');
  END;
 press_any_key;
END;

{————————————————————————————————————}
{  This procedure sets the stack pointers to zero (empty
stack) and }
{  all (pascal) pointers to nil.                      }
{————————————————————————————————————}

PROCEDURE Initialise;

 VAR  t:textual;
   i:whole;

BEGIN
 mark(datastructures);
 memory: = 0;
 bell: = chr(7);
 storetext('I can''t explain any further.', t);
 stackp: = 0;
 stackp2: = 0;
 push(t);

 FOR i: = 1 TO MaxNoOfd DO decision[i]: = NIL;
 FOR i: = 1 TO MaxNoOfq DO question[i]: = NIL;
 FOR i: = 1 TO MaxNoOfr DO rule[i]: = NIL;

 digits: = ['0'..'9'];
 alphabet: = ['A'..'Z'];

 badinput: = FALSE;
 undo: = FALSE;
 not_opFlag: = FALSE;
 clrscr;
END;  { Initialise }
```

```
{-------------------------------------------------------}
{   The main menu allows the user to chose backward chaining, pseudo }
{   forward chaining modes, or to read in a new knowledge base, view }
{   the current knowledge base, or to quit the module.               }
{-------------------------------------------------------}

PROCEDURE MainMenu;

 VAR  choice,choice2   : CHAR;

BEGIN
 REPEAT
  clearscreen;
  textcolor(yellow);
  writeln;writeln;
  writeln('WELCOME TO THE ELECTROCARDIOGRAPH TUTOR');
  writeln;
  writeln;
  textcolor(white);
  writeln(  'Would you like:-' );
  writeln;
  textcolor(yellow);
  write('  1 ');
  textcolor(white);
  writeln('- To choose an interpretation from a list.');
  textcolor(yellow);
  write('  2 ');
  textcolor(white);
  writeln('- Get step by step guidance ');
  textcolor(yellow);
  write('  3');
  textcolor(white);
  writeln(' - Display the set of rules.');
  writeln;
  textcolor(yellow);
  write('  Q');
  textcolor(white);
  writeln(' - Terminate this session (Quit).');
  writeln;
  writeln;
  textcolor(black);
  textbackground(white);
  write('  Press  1, 2, 3,  or Q ');
  textcolor(white);
  textbackground(black);
  choice: = press_one_of(['1','2','3','Q']);
  clearscreen;
  IF badinput AND (choice IN ['1','2'])
   THEN BEGIN
    writeln;
    writeln;
    writeln(bell,'The input has errors and cannot be used.');
    writeln;
    press_any_key;
   END
  ELSE CASE choice OF
   '3':BEGIN
     clearscreen;
     writeln('Would you like to read in a new set of rules (Y/N)');
     choice2: = press_one_of(['Y','N']);
     IF choice2 = 'Y' THEN
      BEGIN
       release(datastructures);
       Initialise;
       ReadInput;
       Audit;
      END
     ELSE dump;
    END;
   '1':User_Validate;
   '2':Be_an_Expert;
   'Q':;
   END;
 UNTIL choice = 'Q';
END;  { main menu }
{-------------------------------------------------------}
PROCEDURE abort{s:string;i:integer};

BEGIN
 writeln;
 writeln(bell,'Problems!   ',bell,'Problems!   ');
 writeln('****** ABORT :',s,' ',i);
 aborted: = TRUE;
 badinput = TRUE;
 press_any_key;
 exit;
END;


{==============================================================================}
{                                                  }
{                    MAIN PROGRAM                  }
{                                                  }
{==============================================================================}


BEGIN  (* main *);
worldmap;
evalactive: = TRUE;
Initialise;
clearscreen;
ReadInput;
Audit;
REPEAT
 aborted: = FALSE;  { allow recovery via exit(MainMenu) }
 MainMenu;
UNTIL  NOT aborted;
clearscreen;
release(datastructures);
END.
```

```
PROGRAM animate;

{————————————————————————}
{    Include Turbo Pascal extended graphics.      }
{————————————————————————}
{$I a:graph.p}


{————————————————————————}
{    Data structures                              }
{————————————————————————}

TYPE buffer = ARRAY[-1000..32306] OF byte;
 picfile = FILE;

VAR name n:STRING[11];
 num:CHAR;
 i,j:INTEGER;
 cur_pic:buffer;
 picfil_var:picfile;

{————————————————————————}
{    Choose default colours and palette           }
{————————————————————————}

PROCEDURE initialise;

BEGIN

 graphColormode;
 palette(2);
 setpencolor(2);
 showturtle;
 home;
 num:='a';
END;

{————————————————————————}
{   Screen painting procedures and functions      }
{————————————————————————}
PROCEDURE drawscreen;

 VAR ch,ch2:CHAR;
  angle,cur_color,X,Y:INTEGER;
  speed:INTEGER;


{————————}
 PROCEDURE drawcircle;

  VAR bucket:CHAR;
   i,j,x1,x2,y1,y2,radius:INTEGER;

BEGIN
 penup;
 showturtle;
 x1:=200;x2:=200;
 y1:=200;y2:=200;
 REPEAT
  read(kbd,bucket);
  IF bucket=#27 THEN
   BEGIN
    read(kbd,bucket);
    CASE bucket OF
    #75:BEGIN
      setheading(270);
      forwd(1);
     END;
    #77:BEGIN
      setheading(90);
      forwd(1);
     END;
    #72:BEGIN
      setheading(0);
      forwd(1);
     END;
    #80:BEGIN
      setheading(180);
      forwd(1);
     END;
    #82:BEGIN
      x1:=xcor+160;
      y1:=99-ycor;
      plot(x1,y1,cur_color);
     END;
    #83:BEGIN
      x2:=xcor+160;
      y2:=99-ycor;
      i:=((x1-x2)*(x1-x2))+(y1-y2)*(y1-y2);
      radius:=trunc(sqrt(abs(i)));
      plot(x1,y1,0);
      circle(x1,y1,radius,cur_color);
     END;
    END;{case}
   END;
  UNTIL bucket=#83;
END;

{————————}
 PROCEDURE deletewindow;

  VAR bucket:CHAR;
   x,y,w,h,x1,x2,x3,y1,y2,y3:INTEGER;
   buffer:ARRAY[0..12000] OF byte;

BEGIN
 penup;
```

```
    showturtle;
    x1:=0;x2:=0;
    y1:=0;y2:=0;
    x1:=xcor+160;
    y1:=99-ycor;
    plot(x1,y1,cur_color);
    REPEAT
     read(kbd,bucket);
     IF bucket=#27 THEN
      BEGIN
       read(kbd,bucket);
       CASE bucket OF
       #59 BEGIN cur_color:=0;setpencolor(cur_color);END;
       #60 BEGIN cur_color:=1;setpencolor(cur_color);END;
       #61 BEGIN cur_color:=2;setpencolor(cur_color);END;
       #62 BEGIN cur_color:=3;setpencolor(cur_color);END;
       #75 BEGIN
         setheading(270);
         forwd(1);
        END;
       #77 BEGIN
         setheading(90);
         forwd(1);
        END;
       #72 BEGIN
         setheading(0);
         forwd(1);
        END;
       #80:BEGIN
         setheading(180);
         forwd(1);
        END;
       #83 BEGIN
         x2:=xcor+160;
         y2:=99-ycor;
         plot(x1,y1,0);
         IF x1x2 THEN BEGIN
           x:=x1;
           x1:=x2;
           x2:=x;
          END
         ELSE IF x1=x2 THEN x2:=x2+1;
         IF y1y2 THEN BEGIN
           y:=y1;
           y1:=y2;
           y2:=y;
          END
         ELSE IF y1=y2 THEN y2:=y2+1;
         graphwindow(x1,y1,x2,y2);
         IF cur_color=0 THEN fillscreen(4)
         ELSE fillscreen(cur_color-1);
         graphwindow(0,0,319,199);
        END;
      END;{case}
    END;
  UNTIL (bucket=#83);
END;
```

```
{————————————}
 PROCEDURE copywindow;

  VAR bucket:CHAR;
   x,y,w,h,x1,x2,x3,y1,y2,y3:INTEGER;
   buffer:ARRAY[0..12000] OF byte;
                          /
 BEGIN
  penup;
  showturtle;
  x1:=0;x2:=0;
  y1:=0;y2:=0;
  x1:=xcor+160;
  y1:=99-ycor;
  plot(x1,y1,cur_color);
  REPEAT
   read(kbd,bucket);
   IF (bucket=#27) AND keypressed THEN
    BEGIN
     read(kbd,bucket);
     CASE bucket OF
     #75:BEGIN
       setheading(270);
       forwd(1);
      END;
     #77:BEGIN
       setheading(90);
       forwd(1);
      END;
     #72:BEGIN
       setheading(0);
       forwd(1);
      END;
     #80:BEGIN
       setheading(180);
       forwd(1);
      END;
     #83:BEGIN
       x2:=xcor+160;
       y2:=99-ycor;
       plot(x1,y1,0);
       IF x1x2 THEN BEGIN
         x:=x1;
         x1:=x2;
         x2:=x;
        END
       ELSE IF x1=x2 THEN x2:=x2+1;
       IF y1y2 THEN BEGIN
         y:=y1;
         y1:=y2;
         y2:=y;
        END
       ELSE IF y1=y2 THEN y2:=y2+1;
       getpic(buffer,x1,y1,x2,y2);
       graphwindow(x1,y1,x2,y2);
       graphwindow(0,0,319,199);
```

```
        END;
       #82:BEGIN
         x3: = xcor + 160;
         y3: = 99-ycor;
         putpic(buffer,x3,y3);
        END;
       END;{case}
     END;
  UNTIL (bucket = #82);
 END;

{————————————————————————}

BEGIN
 angle: = 0;
 cur_color: = 3;
 REPEAT
  read(kbd,ch);
  IF (ch = #27) AND keypressed THEN BEGIN
     read(kbd,ch);
     IF keypressed THEN speed: = 20
     ELSE speed: = 1;
     CASE ch OF
     #73:BEGIN angle: = angle + 4;
       setheading(angle);
      END;
     #71:BEGIN angle: = heading-4;
       setheading(angle);
      END;
     #81:angle: = 0;
     #79:angle: = 0;
     #175:forwd(speed);
     #183:back(speed);
     #72:BEGIN
       setheading(angle);
       forwd(speed);
      END;
     #80:BEGIN
       setheading(angle + 180);
       forwd(speed);
      END;
     #77:BEGIN
       setheading(angle + 90);
       forwd(speed);
      END;
     #75:BEGIN
       setheading(angle + 270);
       forwd(speed);
      END;
     #59:BEGIN cur_color: = 0;setpencolor(cur_color);END;
     #60:BEGIN cur_color: = 1;setpencolor(cur_color);END;
     #61:BEGIN cur_color: = 2;setpencolor(cur_color);END;
     #62:BEGIN cur_color: = 3;setpencolor(cur_color);END;
     #84:BEGIN
       showturtle;
       penup;
       forwd(speed);
       x: = xcor + 159;
       y: = 99-ycor;
       fillshape(x,y,0,cur_color);
       pendown;
      END;
     #85:BEGIN
       showturtle;
       penup;
       forwd(speed);
       x: = xcor + 159;
       y: = 99-ycor;
       fillshape(x,y,1,cur_color);
       pendown;
      END;
     #86:BEGIN
       showturtle;
       penup;
       forwd(speed);
       x: = xcor + 159;
       y: = 99-ycor;
       fillshape(x,y,2,cur_color);
       pendown;
      END;
     #87:BEGIN
       showturtle;
       penup;
       forwd(1);
       x: = xcor + 159;
       y: = 99-ycor;
       fillshape(x,y,3,cur_color);
       pendown;
      END;
     #63:pendown;
     #88:penup;
     #64:drawcircle;
     #82:copywindow;
     #83:deletewindow;
     #89:BEGIN
       colortable(3,2,1,0);
       fillscreen(-1);
       colortable(0,1,2,3);
      END;
     #65:BEGIN
       clearscreen;
       home;
      END;
     #90:palette(cur_color);
     #66:showturtle;
     #166,#15:BEGIN
       clearscreen;
       writeln('enter number');
       read(kbd,ch2);
       name: = 'c:q.' + ch2;
       BEGIN
        assign(picfil_var,name);
        reset(picfil_var);
        blockread(picfil_var,cur_pic,156);
```

```
        close(picfil_var);
        putpic(cur_pic,0,199);
      END;
      drawscreen;
    END;
  #91:hideturtle;
  #68: BEGIN
      getpic(cur_pic,0,0,320,200);
      REPEAT
        writeln('enter number ?',num,' ?');
        read(kbd,num);
      UNTIL (num '0') AND (num ':');
      putpic(cur_pic,0,199);
      name: = 'c:' + n + '.' + num;
      num: = succ(num);
      assign(picfil_var,name);
      rewrite(picfil_var);
      blockwrite(picfil_var,cur_pic,256);
      close(picfil_var);
    END;
  END; {case}
 END;
UNTIL (ch = #93) OR (ch#120);
END;

{————————————————————————}
{          MAIN PROGRAM                }
{————————————————————————}

BEGIN
initialise;
writeln('ENTER NAME FOR PICTURE FILES');
read(n);
IF n'Q' THEN BEGIN

  name: = 'c:' + n + '.' + num;
  clearscreen;
  IF n = 'q' THEN BEGIN
    assign(picfil_var,name);
    reset(picfil_var);
    blockread(picfil_var,cur_pic,156);
    close(picfil_var);
    putpic(cur_pic,0,199);
   END;
  drawscreen;
 END
ELSE BEGIN read(num); n: = 'q';END;
j: = abs(ord(num)-ord('1'));
REPEAT
 num: = '1';
 hideturtle;
 FOR i: = 1 TO j DO BEGIN
    name: = 'c:' + n + '.' + num;
    num: = succ(num);
    assign(picfil_var,name);
    reset(picfil_var);
    blockread(picfil_var,cur_pic,156);
  close(picfil_var);
  putpic(cur_pic,0,199);
 END;
FOR i: = j DOWNTO 1 DO BEGIN
  num: = pred(num);
  name: = 'c:' + n + '.' + num;
  assign(picfil_var,name);
  reset(picfil_var);
  blockread(picfil_var,cur_pic,256);
  close(picfil_var);
  putpic(cur_pic,0,199);
 END;
UNTIL keypressed;
END.
```

```
PROGRAM compress;

{-------------------------------------}
{     Include Turbo Pascal extended graphics     }
{-------------------------------------}

{$I graph.p}
{R+}

{-------------------------------------}
{          Data structures            }
{-------------------------------------}

TYPE data = RECORD
  index:INTEGER;
  value:byte;
 END;

VAR
 picbuf1,picbuf2:ARRAY[1..21306] OF byte;
 picfile      :FILE;
 datafile     :FILE;
 databuf      :ARRAY[0..6000] OF data;
 r1,r2,i,j    :INTEGER;
 k            :INTEGER;
 filename     :STRING[10];
 OK           :BOOLEAN;
 ch,suffix    :CHAR;
 name         :STRING[8];
 realnum      :REAL;


{-------------------------------------}
{     Delta compression algorithm              }
{-------------------------------------}
PROCEDURE compress;

BEGIN clrscr;
 filename:=name+'.'+ch;
 assign(picfile,filename);
   {$I-}reset(picfile);{$I+}
 OK:=(IOresult=0);
 IF NOT OK THEN halt;
 r1:=round(sizeof(picbuf1)/128);
 blockread(picfile,picbuf1,r1);
 close(picfile);


 REPEAT
  ch:=succ(ch);
  filename:=name+'.'+ch;
  assign(picfile,filename);
   {$I-}reset(picfile);{$I+}
  OK:=(IOresult=0);
 IF OK THEN
  BEGIN
   blockread(picfile,picbuf2,r1);
   close(picfile);

   graphmode;

   i:=21306;
   k:=0;
   FOR j:=1 TO i DO
    IF picbuf1[j]picbuf2[j]
     THEN BEGIN
      k:=k+1;
      IF k MOD 500 = 0 THEN writeln(k);
      WITH databuf[k] DO
       BEGIN
        index:=j;
        value:=picbuf2[j];
       END;
     END;
   write('k = ',k);
   databuf[0].index:=(k);
   filename:=name+'dc.'+ch;
   assign(datafile,filename);
   rewrite(datafile);
   realnum:=3*k+3;
   r2:=trunc(1+realnum/128.0);
   databuf[0].value:=r2;
   blockwrite(datafile,databuf,r2);
   close(datafile);
   picbuf1:=picbuf2;
  END;
 UNTIL NOT OK;
END;


{-------------------------------------}
{     Expand and display the compressed pictures     }
{     to check the compression procedure.            }
{-------------------------------------}

PROCEDURE expand;

BEGIN
 clrscr;
 graphmode;
 REPEAT
  ch:=suffix;
  filename:=name+'.'+ch;
  assign(picfile,filename);
  reset(picfile);
  r1:=round(sizeof(picbuf1)/128);
  blockread(picfile,picbuf1,r1);
  close(picfile);
```

```
  putpic(picbuf1,0,199);

  REPEAT
   ch: = succ(ch);
   filename: = name + 'dc.' + ch;
   assign(datafile,filename);
   {$I-}reset(datafile);{$I+}
   OK: = (IOresult = 0);
   IF OK THEN
    BEGIN
     blockread(datafile,databuf,1);
     r2: = databuf[0].value;
     IF databuf[0].index0 THEN
      BEGIN
       IF r21 THEN
        BEGIN
             {$I-}reset(datafile);{$I+}
         OK: = (IOresult = 0);
         blockread(datafile,databuf,r2);
        END;
       close(datafile);

       FOR j: = 1 TO  databuf[0].index DO
        BEGIN
         WITH databuf[j] DO
          picbuf1[index]: = value;
        END;
       putpic(picbuf1,0,199);
       ch: = succ(ch);
      END;
    END;
  UNTIL NOT OK;
 UNTIL keypressed;
 textmode;
END;

{————————————————————————}
{          MAIN PROGRAM                  }
{————————————————————————}

BEGIN
write('Enter name of file ');
read(name);
suffix: = 'a';
ch: = suffix;
compress;
expand;
END.
```

APPENDIX IV

LISTING OF KNOWLEDGE BASES

WOLFF-PARKINSON WHITE-SYNDROME

```
D1 'Ventricular pre-excitation WPW pattern type A'
D2 'Ventricular pre-excitation WPW pattern type B'
D3 'Wolff-Parkinson-White syndrome not present'
D4.

Ques 1
'Are atrial flutter or atrial fibrillation waves present?'
answer 1 'Yes'
2 'No'.

Ques 2
'Are P waves present?'
answer 1 'Yes'
2 'No'.

Ques 3
'Are delta waves present in three or more leads?'
answer 1 'Yes'
2 'No'.

Ques 4
'Is the QRS area in lead V1:-'
answer 1 'Positive'
2 'Negative'
3 'Neither'.

Ques 5
'Is the PR interval in V1:'
answer 1 '> 180 ms'
2 '< 180 ms but > 140 ms'
3 '< 140 ms '.

Rule 1 : IF q1a2 & q2a1 & q3a1 then d4.
Rule 2 : IF d4 & q4a1 & q5a2 then d1.
Rule 3 : IF d4 & q4a2 & q5a3 then d2.
Rule 4 : IF (not d1) & (not d2) then d3.
```

MYOCARDIAL INFARCTS

D1 'Cannot rule out anterior infarct. Possibly acute'
D2 'Possible anterior infarction. Possibly acute'
D3 'Anterior infarction, possibly acute'
D4 'Cannot rule out anterior infarction, age undetermined'
D5 'Possible anterior infarction, age undetermined'
D6 'Anterior infarction, age undetermined'
D7 'Cannot rule out septal infarction. Possibly acute'
D8 'Septal infarction, possibly acute'
D9 'Cannot rule out septal infarction, age undetermined'
D10 'Septal infarction, age undetermined'
D12 'Possible lateral infarct, posibly acute'
D13 'Lateral infarct, possibly acute'
D14 'Possible lateral infarct, age undetermined'
D15 'Lateral infarct, age undetermined'
D16 'Anteriorseptal infarct'
D17 'Anteriolateral infarct'
D18 'Anterioseptal infarct, possibly acute'
D19 'Anterioseptal infarct, age undetermined'
D20 'Cannot rule out inf. infarct masked by l.ant.fasc.block'
D21 'Cannot rule out inferior infarct'
D22 'Possible inferior infarction'
D23 'Inferior infarction, age undetermined'
D24 'Inferior infarction, possibly acute'
D25 'Inferior infarction with posterior extention'
D30
D31
D32
D33
D34
D35
D36
D38
D39
D40
D41
D42
D43
D44
D45
D46
D47
D48
D49
D50
D51
D52
D53
D54
D55
D56
D57
D58
D59
D60
D61.

```
Question 1
'Is the Q wave duration in V3'
answer 1 'less than 20ms'
2 'between 20ms and 30ms'
3 'greater than 30ms'.

Question 2
'Is the Q wave deflection in V3'
answer 1 'absent, no Q wave'
2 'less than 50uV'
3 'between 50uV and 75uV'
4 'between 75uV and 100uV'
5 'greater than 100uV but less than 200uV'
6 'greater than 200uV' .

Question 3
'Is the Q wave duration in V4'
answer 1 'less than 30ms'
2 'greater than 30ms'.

Question 4
'Is the Q wave amplitude in V4'
answer 1 'absent, no Q wave'
2 'less than 75uV'
3 'between 75uV and 100uV'
4 'greater than 100uV'.

Question 5
'Is the QRS complex'
answer 1 'negative in V1'
2 'negative in V2'
3 'negative in both V1 and V2'
4 'none of these'.

Question 6
'Is the QRS complex'
answer 1 'negative in V3'
2 'negative in V4'
3 'negative in both V3 and V4'
4 'none of these'.

Question 8
'Is the R or R' deflection in V3'
answer 1 'less than 25uV the R deflection in V2'
2 'less than the R deflection in V2, but not 25uV less'
3 'greater than the R deflection in V2'.

Question 9
'Is the total QRS complex duration in V3'
answer 1 'less than 50ms'
2 'greater than 50ms'.

Question 10
'Is the R or R' deflection in V3'
answer 1 'less than 100uV'
2 'greater than 100uV but less than 200uV'
3 'greater than 200uV'.
```

```
Question 11
'Is the total QRS deflection in V3'
answer 1 'less than 50uV'
2 'greater than 50uV'.

Question 12
'Is the maximum R amplitude in V4'
answer 1 'less than 200uV'
2 'greater than 200uV'.

Question 13
'Is the R or R' deflection in V4'
answer 1 'less than 25uV than the R amplitude in V3'
2 'less than the R amplitude in V3, but not 25uV less'
3 'greater than the R amplitude in V3'.

Question 14
'Is the total QRS duration in V4'
answer 1 'less than 50ms'
2 'greater than 50ms'.

Question 15
'Is the total QRS deflection in V4'
answer 1 'less than 50uV'
2 'greater than 50uV'.

Question 16
'Is the Q duration in V2'
answer 1 'less than 20ms'
2 'between 20ms and 30ms'
3 'greater than 30ms'.

Question 17
'Is the Q amplitude in V2'
answer 1 'absent - no Q wave'
2 'less than 200uV'
3 'greater than 200uV'.

Question 18
'Is the QRS duration in any lead'
answer 1 'greater than 120ms'
2 'less than 120ms'.

Question 19
'Is this a low voltage E.C.G.?'
answer 1 'Yes'
2 'No'.

Question 20
'Are bundle branch blocks (left or right) present?'
answer 1 'Yes'
2 'No'.

Question 21
'Is the T or T' deflection in V3'
answer 1 'more negative than -100uV (e.g. -120 uV)'
2 'less negative than -100uV'
3 'positive'.
```

```
Question 22
'Is the T or T' deflection in V4'
answer 1 'more negative than -100uV (e.g. -120 uV)'
2 'less negative than -100uV'
3 'positive'.


Question 23
'Is the S-T elevation in lead V3'
answer 1 'greater than 200uV'
2 'less than 200uV'
3 'S-T elevation not present'.

Question 24
'Is the S-T elevation in lead V4'
answer 1 'greater than 200uV'
2 'less than 200uV'
3 'S-T elevation not present'.

Question 25
'Is the R deflection in V2'
answer 1 'more than 50uV less than the R deflection in V1'
2 'less than the deflection in V1, but not 50uV less'
3 'greater than the R deflection in V1'.

Question 26
'Is the R deflection in V2'
answer 1 'greater than 50uV'
2 'less than 50uV'.

Question 28
'Is the Q deflection in V2'
answer 1 'less than 100uV'
2 'between 100uV and 200uV'
3 'greater than 200uV'.

Question 30
'Is the S-T elevation in V1'
answer 1 'greater than 200uV'
2 'less than 200uV'
3 'no elevation'.

Question 31
'Is the S-T elevation in V2'
answer 1 'greater than 200uV'
2 'between 200uV and 50uV'
3 'less than 50uV'
4 'no elevation'.

Question 32
'Are the T waves in V1'
answer 1 'Positive'
2 'Negative'.

Question 33
'Are the T waves in V2'
answer 1 'Positive'
2 'Negative'.
```

```
Question 36
'Is the total QRS deflection in V5'
answer 1 'greater than 50uV'
2 'less than 50uV'.

Question 37
'Is the R or R' deflection in V5'
answer 1 'less than 100uV'
2 'greater than 100uV'.

Question 38
'Is the total QRS duration in V5'
answer 1 'less than 50ms'
2 'greater than 50ms'.

Question 41
'Is the total QRS duration in V6'
answer 1 'less than 50ms'
2 'greater than 50ms'.

Question 42
'Is the QRS amplitude in V6'
answer 1 'less than 50uV'
2 'greater than 50uV'.

Question 43
'Is the Q duration greater than 25 in'
answer 1 'Std lead I'
2 'aVl'
3 'V5'
4 'V6'
5 'some of the above'
6 'all of the above'.

Question 44
'Is the Q deflection greater than 75uV in'
answer 1 'Std lead I'
2 'aVl'
3 'V5'
4 'V6'
5 'some of the above'
6 'all of the above'.

Question 45
'Is the R deflection less than one fifth the Q deflection'
answer 1 'in V6'
2 'in V5'
3 'both V5 and V6'
4 'neither'.

Question 50
'Is the ST segment elevated by more than 100uV in'
answer 1 'Std lead I'
2 'aVl'
3 'V5'
4 'V6'
5 'none of these'.
```

```
Question 51
'Is the T wave positive in'
answer 1 'Std lead I and aVl'
2 'V5 and V6'
3 'both'
4 'neither'.

Question 52
'Is the sum of the Q and R durations in aVf'
answer 1 'less than 20ms'
2 'greater than 20ms'.

Question 53
'In std lead II, is the Q deflection'
answer 1 'less than 50uV'
2 'between 50uV and 75uV'
3 'between 75uV and 100uV'
4 'greater than 100uV'.

Question 54
'In lead aVf, is the Q deflection'
answer 1 'less than 50uV'
2 'between 50uV and 75uV'
3 'between 75uV and 100uV'
4 'greater than 100uV'.

Question 55
'In std lead II, is the Q duration'
answer 1 'less than 20ms'
2 'between 20ms and 25ms'
3 'between 25ms and 30ms'
4 'between 30ms and 35ms'
5 'between 35ms and 40ms'
6 'greater than 40ms'.

Question 56
'In lead aVf, is the Q duration'
answer 1 'less than 20ms'
2 'between 20ms and 25ms'
3 'between 25ms and 30ms'
4 'between 30ms and 35ms'
5 'between 35ms and 40ms'
6 'greater than 40ms'.

Question 57
'Is the Q wave minus the T wave deflection in Std lead II'
answer 1 'greater than one half of the R wave '
2 'greater than one third of the R wave '
3 'greater than one quarter of the R wave '
4 'greater than one fifth of the R wave '
5 'none of these'.

Question 58
'Is the QRS axis'
answer 1 'less than -30 degrees'
2 'between -30 and +30 degrees'
3 'between +30 and +180 degrees'
3 'greater than 180 degrees'.
```

```
Question 59
'Is the ST segment in aVf'
answer 1 'elevated by between 50uV and 100uV'
2 'elevated by more than 100uV'
3 'neither'.

Question 60
'Is the ST segment in Std lead II'
answer 1 'elevated by between 50uV and 100uV'
2 'elevated by more than 100uV'
3 'neither'.

Question 61
'In Std II, is the ST segment ,immediately before the T wave'
answer 1 '200uV or more above the T wave peak'
2 'between 100uV and 200uV'
3 'less than 100uV above the T wave'
4 'T wave not inverted'.

Question 63
'Is the T wave deflection'
answer 1 'negative and more than 50uV in aVf'
2 'negative, and more than 50uV in Std lead II'
3 'both'
4 'neither'.

Question 64
'Is the Q wave in V1'
answer 1 'present'
2 'absent'.

Question 65
'Is the R duration in V1'
answer 1 'greater than 40ms'
2 'between 30ms and 40ms'
3 'less than 30ms'.

Question 66
'Is the QRS deflection in V2'
answer 1 'greater than 50uV'
2 'less than 50uV'.

Question 67
'Is W-P-W syndrome present'
answer 1 'Yes'
2 'No'.

Question 68
'Is left ventricular hypertrophy present'
answer 1 'Yes'
2 'No'.

Question 69
'Is the R or R' deflection in V6'
answer 1 'less than 100uV'
2 'greater than 100uV'.
```

```
Question 70
'Is the Q duration greater than 30ms in'
answer 1 'Std lead I'
2 'aVl'
3 'V5'
4 'V6'
5 'some of the above'
6 'all of the above'.

Question 71
'Is a left facsicular block indicated'
answer 1 'Yes'
2 'No'.

Question 72
'Is the Q wave minus the T wave deflection in aVf'
answer 1 'greater than one half of the R wave '
2 'greater than one third of the R wave '
3 'greater than one quarter of the R wave '
4 'greater than one fifth of the R wave '
5 'none of these'.

Question 73
'Is the T axis greater than 180 degrees'
answer 1 'Yes'
2 'No'.

Question 74
'In aVf, is the ST segment ,immediately before the T wave'
answer 1 '200uV or more above the T wave peak'
2 'between 100uV and 200uV'
3 'less than 100uV above the T wave'
4 'T wave not inverted'.

Question 75
'Is the R duration in V2'
answer 1 'greater than 40ms'
2 'between 30ms and 40ms'
3 'less than 30ms'.


Rule 1 : IF (q1a3 & (q2a3 or q2a4 or q2a4 or q2a6)) or
            (q3a2 & (q4a2 or q4a4)) then d30.
Rule 2 : IF ((q2a5 or q2a6) & (q6a1 or q6a3)) or
            (q3a4 & (q4a3 or q4a4)) then d31.
Rule 3 : IF (q5a3 & q68a2 & q8a1 & (not q10a3) & q9a1 & q11a2) or
            ((not  q10a3) & q8a1 & q6a3 & q68a2 & q11a2  &  q9a2)
            then d32.
Rule  4  :  IF q5a3 & q68a2 & q12a1 & q13a1 &q15a2  &  (q14a1  or
               q14a2) then d33.
Rule 5 : IF q16a2 & (not q1a1) & q2a6 & q17a3 then d34.
Rule  6  : IF q68a2 & q18a2 & q17a1 & q10a1 & q11a2  &  (q9a1  or
              q9a2) then d35.
Rule 7 : IF (q23a1 or q24a1) & q18a2 & q21a3 & q22a3 then d36.
Rule  8 : IF (D30 or D31 or D32 or D33 or D34 or D35) & D36  then
              d1.
Rule  9 : IF (d30 or d31 or d32 or d33 or d34 or d35) &  not  d36
            then d4.
Rule 10 : IF (d30 or d31 or d32 or d33 or d34 or d35) & (not q2a1) &
             q19a2 & q20a2 & d36 then d2.
```

```
Rule 11 : IF (d30 or d31 or d32 or d33 or d34 or d35) & (not q2a1) &
          (not q4a1) & q19a2 & q20a2 & (not d36) then d5.
Rule  12 : IF (d30 or d31 or d32 or d33 or d34 or d35) & q12a2  &
           q20a2 & ((q21a1 or q22a1) or (q1a3 or q3a3)) &  d36
           then d3.
Rule  13 : IF (d30 or d31 or d32 or d33 or d34 or d35) & q12a2  &
           q20a2  &  ((q21a1 or q22a1) or (q1a3 or  q3a3))  &
           (not d36) then d6.
Rule 14 : IF q67a2 & q20a2 then d38.
Rule 15 : IF q25a1 & q26a2 & (not (d1 or d2)) & q66a1 & d38 then d39.
Rule 16 : IF q16a3 & d38 then d40.
Rule 17 : IF (not q28a1) & ((q5a2 or q5a3) or q20a1) & d38 then d41.
Rule 18 : IF (q30a1 or q31a1) & q32a1 & q33a1 then d42.
Rule 19 : IF (d39 or d40 or d41) then d9.
Rule 20 : IF d9 & ((q31a1 or q32a2) or (q20a2 & q68a2)) & d42 then d8
Rule  21 : IF d9 & ((q31a1 or q32a2) or (q20a2 & q68a2))  &  (not
          d42) then d10.
Rule 22 : IF q67a2 & q38a2 & q36a1 & q37a2 &
          (not ((q1a3 or q3a3) & ((q2a4 or q2a5 or q2a6) or (q4a3
          or  q4a4))) & (not ((q1a3 or q3a3) & ((q2a4 or q2a5  or
          q2a6)  or (q4a4 & (q2a2 or q2a3))))) & (not (((q2a5  or
          q2a6) & (q6a1 or q6a3)) or (q4a4 & (q2a2 or q2a3)))
          & (not ((not q1a1) & (not q16a1) & q17a3 & q2a6 & q37a1
          &  q41a2 & q42a2) or ( q41a2 & q42a2 &  q69a2)))) then
          d43.
Rule  23 : IF (q69a1 & q41a2 & q42a2) or (q41a2 & q42a2 &  q69a2)
           then d44.
Rule 24 : IF (q43a5 or q43a6) then d45.
Rule 25 : IF (q44a5 or q44a6) then d46.
Rule 26 : IF not q45a4 then d47.
Rule 27 : IF not q45a3 then d48.
Rule 28 : IF (d45 & (d46 or d47 or d48)) or (d46 & (d47 or  d48))
          or (d47 & d48) then d49.
Rule 29 : IF (not q50a5) & q51a3 & (not q18a1) then d50.
Rule 30 : IF (d43 or d44 or d49) & d50 then d12.
Rule 31 : IF (d43 or d44 or d49) & not d50 then d14.
Rule 32 : IF (((q70a5 or q70a6) & (d46 or d47 or d48)) or (d46  &
          (d47 or d48))) & d50 then d13.
Rule 33 : IF (((q70a5 or q70a6) & (d46 or d47 or d48)) or (d46  &
          (d47 or d48)))& (not d50) then d15.
Rule 34 : IF (d1 or d2 or d3 or d4 or d5 or d6) & (d7 or d8 or d9
          or d10)& (not (d12 or d13 or d14 or d15)) then d16.
Rule 35 : IF (d1 or d2 or d3 or d4 or d5 or d6) & (d12 or d13  or
          d14 or d15) then d17.
Rule 36 : IF (d4 or d5 or d6) & (d9 or d10) & d16 then d19.
Rule 37 : IF d16 & (not d19) then d18.
Rule 38 : IF q52a1 & q71a1 then d20.
Rule 39 : IF (not (q59a2 or q60a2) & q63a4 & q18a1) then d51.
Rule 40 : IF ((not q53a1) & not(q55a1 or q55a2) & q57a4) or
             ((not q54a1) & (not (q56a1 or q56a2)) & q72a4) or
             (not q53a1 & not q55a1 & (q58a1 or q58a4)) or
             ((q58a1 or q58a2) & q73a1) then d21.
Rule 41 : IF ((q53a3 or q53a4) & (q55a5 or q55a6) & q57a4) or
             ((q54a3 or q54a4) & (q56a5 or q56a6) & q72a3) then d52.
Rule 42 : IF ((q53a3 or q53a4) & not (q55a1 or q55a2) & q57a3) or
             ((q54a3  or q54a4) & not (q56a1 or q56a2)  &  q72a3)
            then d53.
Rule 43 : IF ((q53a3 or q53a4) & not (q55a1) & q57a2) or
             ((q54a3 or q54a4) & not (q56a1) & q72a2) then d54.
Rule 44 : IF ((q53a3 or q53a4) & q55a1 & (q63a2 or q63a3) or
```

((q54a3 or q54a4) & q56a1 & q56a1 & (q63a1 or q63a3))) then d55.

Rule 45 : IF ((q53a3 or q53a4) & q55a1 & (q60a2) & (q61a2 or q61a3) or ((q54a3 or q54a4) & q56a1 & q59a2)) then d56.

Rule 46 : IF d21 & (d52 or d53 or d54 or d55 or d56) then d22.

Rule 47 : IF d22 & ((q60a2 & q61a1) or (q59a2 & q74a1)) then d57.

Rule 48 : IF (q53a4 & q55a6 & (q57a1 or q57a2 or q58a3)) or (q54a4 & q56a6 & (q72a1 or q72a2 or q72a3)) then d58.

Rule 49 : IF (q53a4 & (q55a6 or q55a5) & (q57a1 or q57a2) or (q54a4 & (q56a6 or q56a5) & (q72a1 or q72a2))) then d59.

Rule 50 : IF (q53a4 & (q55a6 or q55a5 or q55a4) & q57a1) or (q54a4 & (q56a6 or q56a5 or q56a4) & q72a1) then d60.

Rule 51 : IF (q59a2 or q60a2) & q63a4 & q18a1 then d61.

Rule 52 : IF (d57 or d58 or d59 or d60) then d21.

Rule 53 : IF d21 & q20a2 & d61 then d24.

Rule 54 : IF d21 & q20a2 & not d61 then d23.

Rule 55 : IF q67a2 & (d21 & (q17a1 or q64a2) & q18a1 & q20a2) or ((q65a1 or q65a2) & (q75a1 or q75a2) & q5a4) then d25.